The University of Jordan
Authorization Form

I, Esam Y. AL-Nsour authorize the University of Jordan to supply copies of my Thesis/Dissertation to libraries or establishments or individuals on request.

Signature:
Date: 18/7/2011

الجامعة الأردنيــة
كلية الدراسات العليا

التاريـــخ: / /

نموذج رقم (١٨)
اقرار والتزام بقوانين الجامعة الأردنية وأنظمتها
وتعليماتها لطلبة الماجستير

أنا الطالب: عصام عوض عبدالله السرور الرقم الجامعي: ٨٠٨٠٢١٢
التخصص: علم الحاسوب الكلية: الملك عبدالله الثاني لتكنولوجيا المعلومات

اعلن بأنني قد التزمت بقوانين الجامعة الأردنية وأنظمتها وتعليماتها وقراراتها السارية المفعول المتعلقة باعداد رسائل الماجستير والدكتوراة عندما قمت شخصيا" باعداد رسالتي / اطروحتي بعنوان: التعيين على طريقة فصل المصدر العمياء في الشكل التنظيمي الشجري من نوع (K)
....
....
....

وذلك بما ينسجم مع الأمانة العلمية المتعارف عليها في كتابة الرسائل والأطاريح العلمية. كما أنني أعلن بأن رسالتي /اطروحتي هذه غير منقولة أو مستلة من رسائل أو أطاريح أو كتب أو أبحاث أو أي منشورات علمية تم نشرها أو تخزينها في أي وسيلة اعلامية، وتأسيسا" على ما تقدم فانني أتحمل المسؤولية بأنواعها كافة فيما لو تبين غير ذلك بما فيه حق مجلس العمداء في الجامعة الأردنية بالغاء قرار منحي الدرجة العلمية التي حصلت عليها وسحب شهادة التخرج مني بعد صدورها دون أن يكون لي أي حق في التظلم أو الاعتراض أو الطعن بأي صورة كانت في القرار الصادر عن مجلس العمداء بهذا الصدد.

توقيع الطالب: [signature] التاريخ: ١٧ / ٧ / ٢٠١١

An improved node splitting algorithm in R-tree

By

Esam Y. A. Al-Nsour

Supervisor

Dr. Azzam Sleit

This Thesis was submitted in Partial Fulfillment of the Requirements for the Master's Degree of Science in Computer Science.

Faculty of Graduate Studies

The University of Jordan

July, 2011

Committee Decision

This Thesis (An improved node splitting algorithm in R-tree) was successfully Defended and Approved on 10/7/2011.

Examination Committee	Signature
Dr. Azzam Sleit, (Supervisor) Assoc. Prof of Data retrieval and Imaging Data Bases	
Dr. Abdel Latif Abu-Dalhoum, Assoc. Prof of Genetic Algorithms.	
Dr. Hazem Al Hiary Assist. Prof of Image Processing	
Dr. Mohammad A. Abbadi Assoc. Prof of Software and Systems. (Mutah University)	

DEDICATION

To my precious Wife and Kids,

To my Brothers and Sisters,

For your care, support and prayers.

ACKNOWLEDGMENT

I would like to thank deeply my supervisor Dr. Azzam Sleit for his guides, assistance and support which were very important in every step of this research.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CA	California stream file
CAD	Computer Aided Design
CBS	Corner Based Splitting algorithm
CPU	Central Processing Unit
CQ	Containment Query
DB	Data Base
DBMS	Data Base Management System
GB	Giga Byte
GHz	Giga Hertz
GIS	Geographical Information Systems
h	Hilbert value
I/O	Input/output Operation
IQ	Intersection Query
LB	Long Beach stream file
LHV	Largest Hilbert Value
M	Maximum number of objects (entries) in one node
M	Minimum number of objects (entries) in one node
Max fill	Maximum fill value of entries in a single node
MBR	Minimum Bounding Rectangle
Min fill	Minimum fill value of entries in a single node
MMIS	Multi Media Information Systems
NE	North East data file
O	Complexity Order
ObjId	Object's identifier
P	Pointer to a child in a node

PAMs	Point Access Methods
PQ	Point query
RAM	Random Access Memory
SAMs	Spatial Access Methods
SDBMS	spatial data base management system
SDT	Spatial Data Types
WQ	Window Query

An improved node splitting algorithm in R-tree

By

Esam Y. A. Al-Nsour

Supervisor

Dr. Azzam Sleit

ABSTRACT

The R-tree spatial index structure is widely accepted and is used in many spatial applications. Minimizing the overlap and the coverage of the index nodes' MBRs are crucial to the overall performance of the index. Good splits produce an efficient R-tree structure which has a minimal height, minimal overlap between nodes and minimal coverage in each node. Enhancing the node splitting algorithm of the R-tree index reduces the time to construct the index, increases the overall performance and enhances the distribution of data after splitting.

This thesis introduces a new improved method to split over flown nodes of the R-tree index structure called Corner Based Splitting (CBS) algorithm. The CBS algorithm selects the splitting axis which will produce the most even split according to the number of objects, the least coverage area and overlap between the output nodes. This is done by using the distance from each object's center to the nearest corner of the node's MBR to discover how objects are spread inside the node. A list of near objects for each corner is maintained, the count of objects in each list will determine which corners are to be joined to produce the output nodes. If one of the resulting nodes is to receive fewer entries than the Min Fill constraint, a balancing mechanism is invoked to move some of the nearest objects to the splitting axis from the more filled node to the less filled node. The proposed method works for 2 or higher dimensional data as well.

The experiments to test the CBS algorithm using both synthetic and real data test files showed a good improvement in performance for the CBS algorithm over two other splitting algorithms: The original R-tree Quad splitting algorithm and one of the most recent proposed algorithms; the Lowxy splitting algorithm. The CBS algorithm outperforms both algorithms in the index creation time, overlap ratio and total coverage area. The data retrieval tests showed that the CBS algorithm needed less disk accesses (I/O) - in most of the cases – than both algorithms. The improvement percentage for the CBS algorithm over the Quad algorithm reaches up to 23%. The improvement percentage for the CBS algorithm over the Lowxy algorithm reaches up to 37%.

CHAPTER 1

INTRODUTION

1.1 Problem Definition

Spatial database systems are becoming more and more popular in recent years. Examples of such applications are: Computer Aided Design (CAD), Geographical Information Systems (GIS) and multimedia information systems (MMIS). Spatial databases are commercially implemented in various software applications such as Oracle Universal server and Informix spatial data-blades (Shekhar, et al., 1999).

Spatial data objects such as shapes, lines and points often cover areas in multi-dimensional spaces. Therefore, classical one-dimensional database indexing structures like B-tree are inadequate to handle multi-dimensional spatial data indexing or searching. Indices are required for efficient access to spatial data (Manolopoulos, et al., 2006).

The R-tree proposed in (Guttman, 1984) is an access method / indexing technique for multi-dimensional spatial data. R-tree index structure is a height-balanced spatial data index, in which nodes of the tree resemble disk pages. R-tree is capable of indexing multi-dimensional (spatial) objects and handling data representing, storing, and retrieval. R-trees and its variants showed powerful mechanisms and typically are the preferred method for indexing spatial data objects. It has been widely used to index the spatial objects in numerous applications (Manolopoulos, et al., 2006).

An R-tree index record in its leaf nodes contains pointers to the data objects. Objects are grouped using Minimum Bounding Rectangles (MBRs) and are added to the MBR within the index that will lead to the smallest increase in its size. Adding a new entry to a full leaf node in R-tree requires splitting that node into two nodes. Node splitting is critical process for the overall performance of the access method since it determines the

final shape of the structure (Fu et al., 2002). There is a need for efficient and optimal splitting policies which runs at reasonable time complexity.

1.2 Research Objectives

Enhancing the full leaf node splitting algorithm of the R-tree spatial index will reduce the time needed to construct the index and increase the overall performance of the spatial data index. Good splits produce an efficient R-tree structure which has a minimal height, minimal overlap between nodes and minimal coverage in each node. Additionally, an enhanced splitting algorithm will reduce the time required to split a node and enhances the distribution of data after splitting. Most importantly it will support a better query performance.

In this thesis we will introduce a new improved method to split over flown nodes of the R-tree index structure. The new method will be investigated and tested against some other splitting methods in the literature.

1.3 Research Methodology

In this research, we are going to study and evaluate the R-tree index structure and its variants in general and some splitting methods exist in the literature. Hence, proposing a new splitting method that will overcome some of the disadvantages of the existing techniques and improve the overall performance of the original R-tree index structure. The design of the new splitting method is done while taking into consideration the cost-time complexity to be as minimum as possible while paying attention to the importance of the ease of implementation.

We are going to perform an evaluation study to the proposed method. The evaluation study will include the following:

- Study and evaluate the proposed algorithms related to node splitting in R-tree.
- Select synthetic and real life data sets to be used for experimental work.

- Perform experimental study to demonstrate the merit of the proposed algorithm against some existing splitting methods.

1.4 Contribution

This thesis presents a new splitting algorithm named Corner Based Splitting algorithm (CBS). The CBS algorithm splits an R-tree over flown node during insertion into two nodes in a way that minimizes the coverage area and overlap area between the output nodes, while having a minimal time complexity and easy implementation. The CBS algorithm show good results in improving the query performance of the R-tree index compared against the original quadratic-cost algorithm of the R-tree and against one of the most recently published algorithms.

1.5 Thesis Outline

The rest of this thesis is structured as follows:

Chapter 2 reviews the literature of the spatial data types, operations and methods used to index spatial databases. This chapter presents R-tree specifications, characteristics and operations. It will also list some of the R-tree variations and their specifications.

Chapter 3 presents the proposed splitting method; the Corner Based Splitting Algorithm (CBS).

Chapter 4 presents the performance study conducted to the proposed algorithm, query performance tests to the CBS algorithm compared against two other splitting algorithms using synthetic and real data sets, lists the results using tables and figures.

Chapter 5 lists the conclusions and discusses the avenues for future work.

Chapter 2

Literature Work

A database (DB) is a collection of information in digital form managed by a database management system (DBMS). DBMS is software that enables accessing the DB. Databases are powerful in managing (representing, storing and retrieving) data through the usage of indices. The most popular indices for traditional databases are the B-tree and its variants. B-tree (Comer, 1979) and its variants is a widely accepted index either in the academic or commercial applications. Although B-tree is a powerful index, it is a single-dimensional index; traditional indices are incapable of managing multi-dimensional (spatial) data (Manolopoulos, et al., 2006).

The term spatial refers to the fact that objects exist or occur in space. Spatial data include topological, weather, geographical, temporal and directional data (Samet, 1995). A spatial database is a database with added features to handle multi dimensional objects. It is optimized to store and query data related to objects in space such as points, lines and polygons. The Spatial Data Base Management System (SDBMS) is designed to provide spatial functionality like storage and manipulations while providing the traditional functionality of the DBMS. Manipulating spatial data objects like points, lines and shapes in multi-dimensional space requires an index which is not only able to represent and store spatial data, but also supports the spatial relationships and operators eventually provides fast access and high data availability (Guttman, 1984), (Güting, 1994).

The need for representing and manipulating spatial data objects has urged many proposed improvements to the B-tree in order to make it capable of indexing multi-dimensional objects, but none were widely accepted as being capable of supporting

spatial relationships and operators, or representing multi-dimensional objects (Manolopoulos, et al., 2006). Therefore new indexing methods were introduced to fulfill the need for a robust index to multi-dimensional objects. The proposed methods can be categorized into two main categories. One category that indexes multidimensional points, such as Grid File (Nievergelt, et al., 1981), hB-tree (Lomet and Salzberg, 1990), Buddy Tree (Seeger and Kriegel, 1990) and BV-tree (Freeston, 1995). And the second category which indexes multidimensional regions such as: Quadtree (Samet and Webber, 1985) and R-tree (Guttman, 1984).

The following sections try to cover the most important areas in the literature that are relative to our work. Section 1 gives a brief description of the spatial data types, operators and some examples of spatial indices. Section 2 covers the R-tree spatial index and its operations. Section 3 covers some of the R-tree variants and splitting methods exists in the literature.

2.1 Spatial data types, operations and indexing

2.1.1 Spatial Data Types (SDTs): Unlike the alphanumeric data objects which are considered to be simple structures and require simple operations on them, spatial data have specifications related to space (i.e. multi-dimensional). Thus, spatial objects are expected to be more complex which require operations with special semantics to access them (Nievergelt, 1989).

The definition of Spatial Data Types (SDTs) is, to a large degree, responsible for a successful design of spatial data models and the performance of spatial database and spatial query languages. SDTs such as points, lines, regions and some more complex types like partitions and graphs (networks), are needed to model and represent geometric data in database systems. The definition and implementation of SDTs is fundamental to the development of spatial data base management systems (SDBMS),

since it provides a fundamental abstraction for modeling the geometric structure of objects in space along with their relationships, properties and operations. The definition of SDTs should be independent of the data model used by a SDBMS (Schneider, 1997). Spatial applications have different definitions and types of spatial data, examples of spatial data definitions are the ROSE algebra (Guting and Schneider, 1993), and the Open Geographic Information System (Open GIS consortium, www.opengis.org/techno/specs.htm).

Examples of Spatial Data Types are:

- Points: Center of a city, oil well, service station…etc.
- Lines: River, road, railway…etc.
- Regions: Forest, air field, lake…etc.
- Partitions: Countries, districts, land ownership…etc.
- Graphs (Networks): Public transport, electrical power supply, land phones…etc.

Figure 2.1 shows some examples of the Spatial Data Types (SDTs).

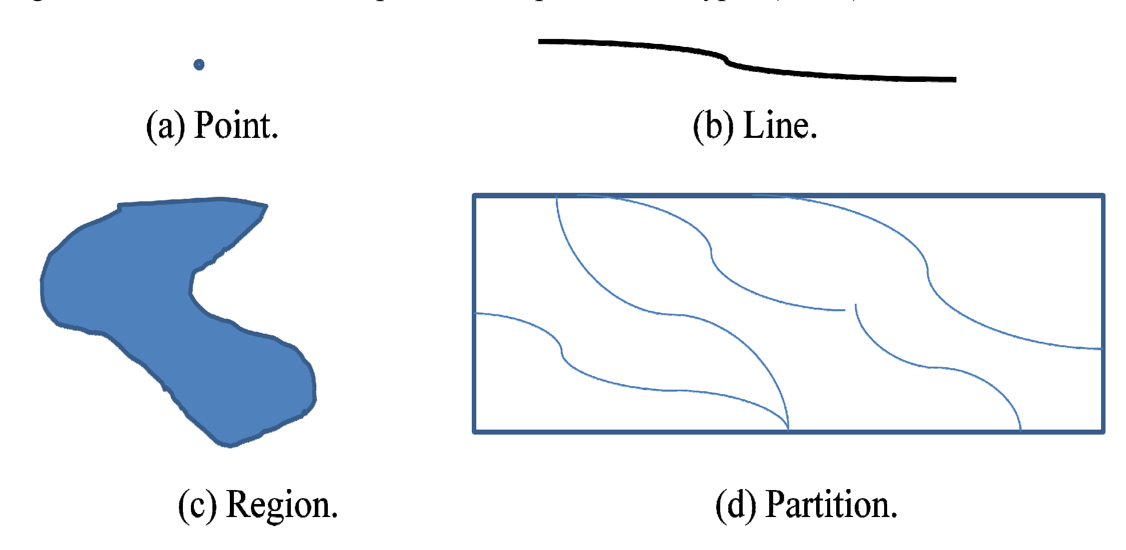


Figure 2.1: Examples of Spatial Data Types.

2.1.2 Spatial Operations (Queries): Spatial operations involve the geometry of objects and their spatial relations (in particular, topological queries are based on topological properties). Therefore, it is important to have data structures that support

efficient computation of geometric properties and spatial relations. Here we list some of spatial queries types. Other types of spatial queries also exist, such as: Adjacency Query (i.e., share a bounding entity with a given object), Nearest-Neighbour Query (find the object that is closest to a given object) and Spatial Join Query (find all pairs of objects that are within a given distance from each other), (Gaede and Günther, 1997):

- Point query: Given a point *P*, find all objects (*o*) that overlap point (*P*), point query is a special type of some other queries: $PQ\ (p) = \{o \mid p \cap o.G = p\}$.
- Window Query (Range Query): Given a d-dimensional interval (I^d), find all objects (*o*) that intersect (I^d). When (I^d) has an arbitrary shape, the query is called Range Query: $WQ\ (I^d) = \{o \mid I^d \cap o.G \neq \emptyset\}$.
- Intersection Query *IQ* (Region Query, Overlap Query): Given an object (*o'*), find all objects (*o*) have at least one point in common with *O'*: $IQ\ (o') = \{o \mid o'.G \cap o.G \neq \emptyset\}$.
- Enclosure Query: Given an object (*o'*), find all objects (*o*) enclosing (*o'*) (completely contained in *o*): $EQ\ (o') = \{o \mid (o'.G \cap o.G) = o'.G\}$.
- Containment Query: Given a object *o'*, find all objects o enclosed by o':

  $CQ\ (o') = \{o \mid (o'.G \cap o.G) = o.G\}$.

The Enclosure Query and the Containment Query have the same formula but with a different result of the intersection. Figure 2.2 shows examples for some spatial queries. Query processing in high-dimensional spaces is fully investigated in (Böhm et al, 2001).

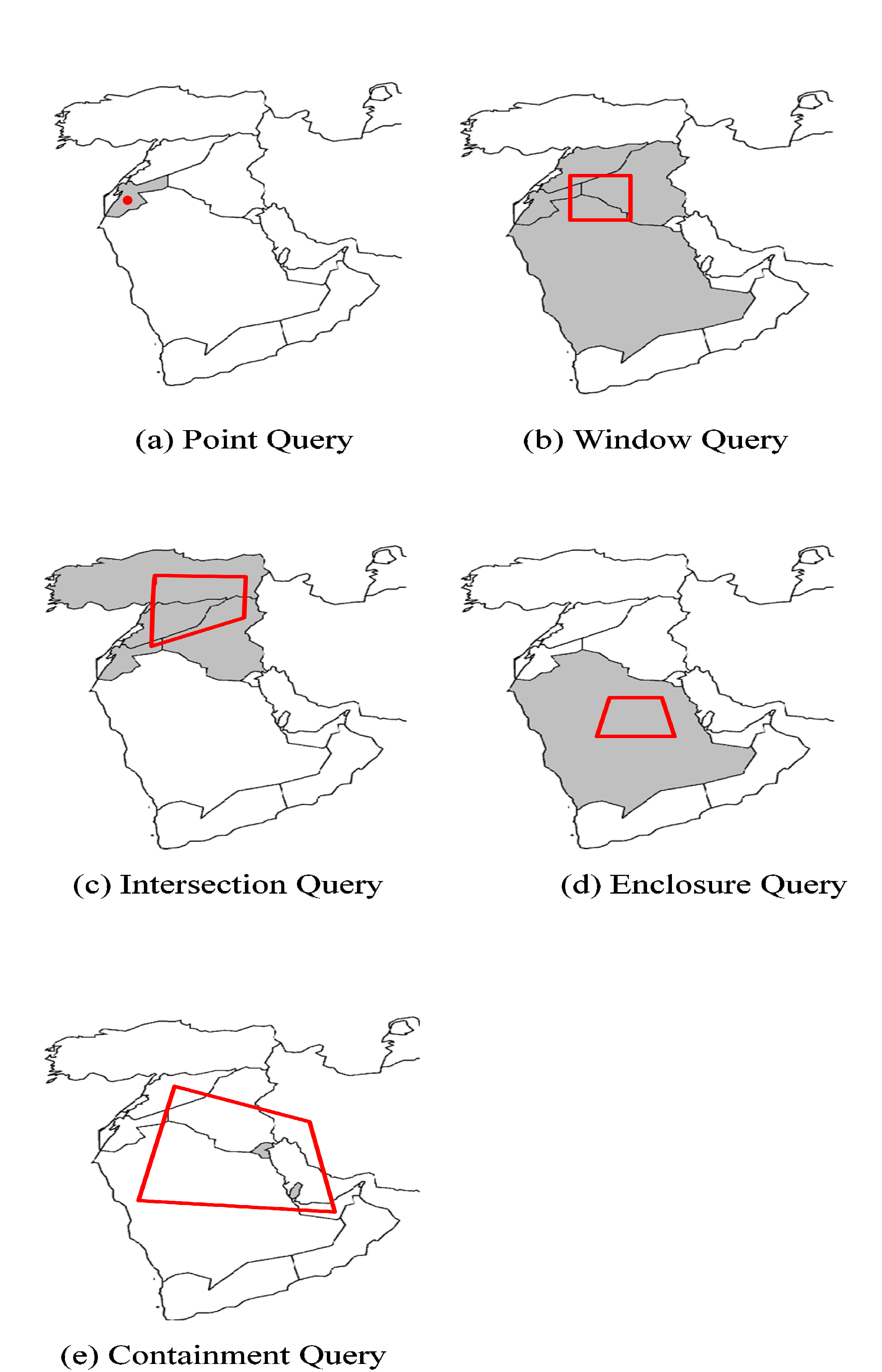


Figure 2.2: Examples of Spatial Query Types.

2.1.3 Spatial Indexing: Specialized multi-dimensional access methods are needed to index multi-dimensional (spatial) data objects. The traditional indexing techniques like B-trees are not adequate for indexing spatial data due to its lack of total ordering, which is an inherent characteristic of a multi-dimensional space. A spatial index is a data-structure; designed to enable fast access to spatial data within the data-store of a spatial database which stores information related to objects in n-dimensional space (Silberschatz, et al., 2006).

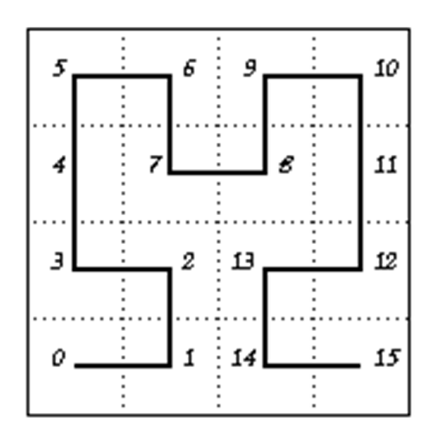
The design of an index structures for spatial data should address basic issues to meet a variety of requirements based on the spatial data properties and their applications. These requirements include: Simplicity, scalability, storage-utilization and space and time efficiency (Gaede and Günther, 1997). In addition, a spatial index should be independent of the type and the insertion sequence of the data, provides fast information retrieval, maintains minimal node area and ensures minimal overlapping between nodes with the least number of objects duplicated to avoid overlap. Other factors such as buffer size, design strategies, space allocation, concurrency and recovery control methods and minimum integration impact of the index into a database system can also affect the performance of spatial information processing. Another factor is the data dimensionality; it affects and deteriorates the query performance (Skopal et al., 2007). A straight-forward solution that fulfills all of these issues is not available.

The "multi-dimensional access methods" term stands for a large class of access methods that support searches in spatial databases. These methods can be categorized into two classes: Point Access Methods (PAMs) and Spatial Access Methods (SAMs), (Gaede and Günther, 1997). A performance comparison of PAMs and SAMs can be found in (Kriegel, et al., 1989).

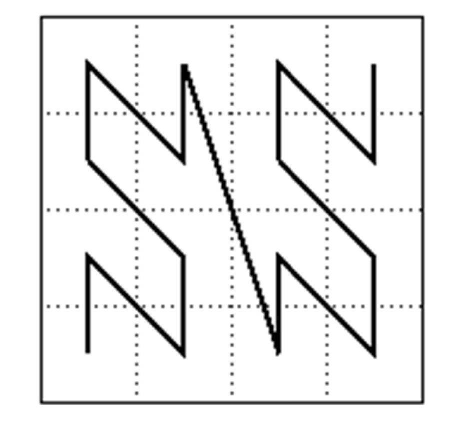
- Point Access Methods (PAM): Designed to handle sets of data points and support spatial searches on them. None of those methods is directly applicable to databases containing objects with a spatial extension. They search for sets of points in two or more dimensions, but they do not have a spatial extension. Points in the database are organized in a number of buckets, each of which corresponds to a disk page and to some subspace of the universe. Buckets are accessed by means of a search tree or some d-dimensional hash function. PAM can be classified into two categories:

➢ Multi-dimensional Hashing Methods: Use one-dimensional hashing to represent the multidimensional objects using different heuristics to preserve the spatial proximity of the objects such as the grid file method (Nievergelt, et al., 1981).

➢ Hierarchical Access Methods: Use hierarchical data structures to store the point data. Hierarchical Methods like Quadtree (Samet, 1984), (Samet and Webber, 1985), K-D-Tree (Bentley, 1975) and K-D-B-Tree (Robinson, 1981). The space filling curves (Figure 2.3) are used to preserve the spatial proximity during the linear ordering of the spatial objects (Lawder and King, 2000). The UB-Tree (Bayer, 1997) uses z-ordering for mapping objects into one-dimensional sequence. The important advantage of the space filling curves is that everything is mapped into one-dimensional space. They are practically insensitive to the number of dimensions if the one-dimensional keys can be arbitrarily large. An obvious disadvantage of space-filling curves is that incompatible index partitions cannot be joined without re computing the codes of at least one of the two indices.



(a) Hilbert curve for 4x4 grid



(b) Z-Order curve for 4x4 grid

Figure 2.3: Space filling curves.

- Spatial Access Methods (SAM): These methods perform operations on spatial objects like: Lines, Polygon and Higher-dimensional objects. They can be considered as an extension of PAM for processing objects with spatial extent. Spatial access methods are a modification of the point access methods using one of the following techniques which can be considered as a classification categories for SAM methods (Seeger and Kriegel, 1988):

➢ Clipping Methods (object duplication): Hierarchical data structures are used as in the case of Overlapping Methods, but these methods clip the objects to prevent overlapping of objects. This ensures that there exists only one path during the search of a data object. The R+-Tree (Sellis, et al., 1987) is an example of this method.

➢ Overlapping Methods (object bounding): Partitioning the data space hierarchically into smaller subspaces. Data objects are stored in the leaf nodes while the intermediate nodes facilitate efficiency during search operations. A node may overlap with its sibling nodes and hence multiple paths may have to be traversed during the course of searching an object. R-Tree and its variants, SS-Tree (White and Jain, 1996), X-Tree (Berchtold, 1996), and SR-Tree (Katayama and Satoh, 1997) are examples of these methods.

- Transformation Methods (object mapping): Mapping spatial objects to points in high-dimensional spaces, then the points are stored using existing PAM. But this approach does not preserve the spatial proximity as the dimensionality increases. Another approach is to use space-filling curves to map objects of higher-dimensional space to one-dimensional points such as Hilbert-R-Tree (Kamel and Faloutsos, 1994) which uses Hilbert-curve in organizing the data.
- Multiple Layers Method: This method can be viewed as a variant of the overlapping method since data regions of different layers may overlap. However, the layers are organized in a hierarchy where each layer partitions the complete universe in a different way. Data regions in the same layer are disjoint and the data regions do not adapt to the spatial extensions of the corresponding data objects. Examples of this method are The Multilayer Grid File (Six and Widmayer, 1988) and The R-File (Hutflesz, et al., 1990).

2.2 R-trees

2.2.1 R-trees Overview: R-tree index structure proposed by Guttman (1984) is a height-balanced spatial data index similar to the one-dimensional B-trees. R-tree is capable of indexing multi-dimensional (spatial) object and handling data representing, storing and retrieval. R-tree and its variants showed powerful mechanisms and typically are the preferred method for indexing spatial data. It is a dynamic index where the insertions and deletions can be intermixed with no periodic reorganization required. Spatial objects (shapes, lines and points) are stored in the leaf nodes, which correspond to disk pages, by using its minimum bounding rectangle (MBR) because they need only two points for their representation. Figure 2.4 shows how objects are stored using its MBR. Although using MBRs to represent objects has some disadvantages, they are the most common used approximation in spatial applications (Papadias, et al., 1995).

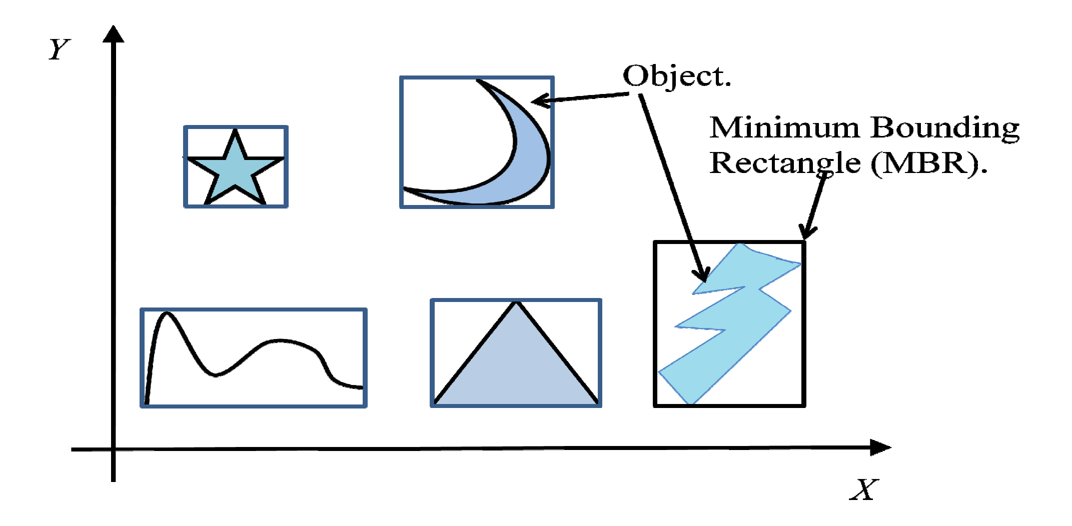


Figure 2.4: Examples of Objects Minimum Bounding Rectangles.

Objects residing in one node are grouped using the MBR of that node. Objects are added to the node that will get the least increase in its MBR size. Each node of the R-tree corresponds to the MBR that bounds its children. The leaves of the tree contain pointers to the database objects instead of pointers to children nodes. Although an MBR can be attached to only one node; MBRs may overlap. This overlapping results in forcing a spatial search to visit many nodes before getting a result. Figure 2.5 shows a case where two spatial objects do not intersect each other, yet their MBRs do. Therefore, the R-tree plays the role of a filtering mechanism to reduce the costly direct examination of geometric objects.

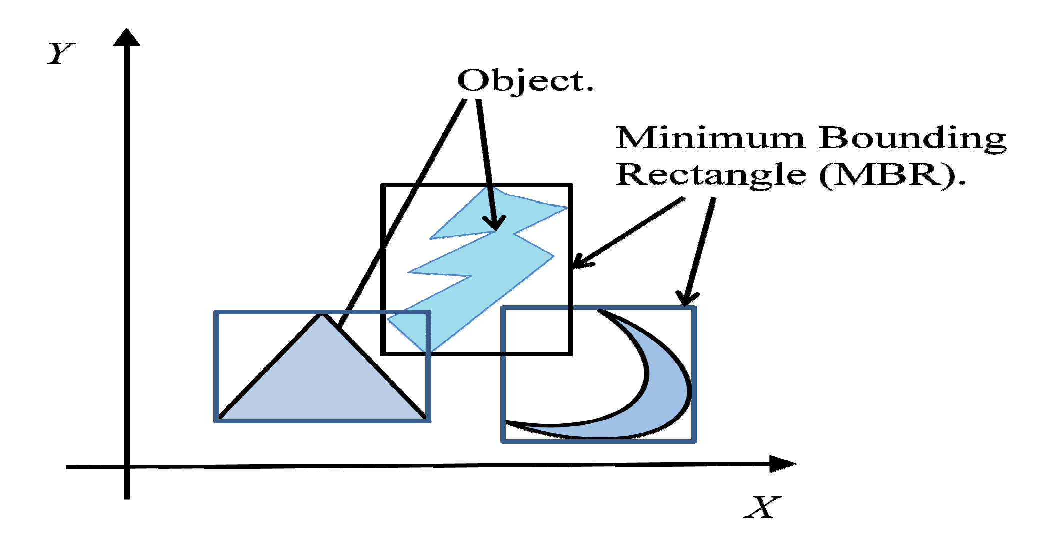


Figure 2.5: Example of intersecting MBRs of non-intersecting objects.

2.2.2 R-tree characteristics: An R-tree of order (*m,M*), where *M* is the maximum number of objects (entries) that will fit in one node while *m* is the minimum number of objects in a node, *m=<M/2*. R-tree satisfies the following properties:

- Every leaf node can host between *m* and *M* entries unless it is the root. Each entry is of the form (*MBR, ObjId*), such that *MBR* is the rectangle that spatially contains the object, *ObjId* is the object's identifier.
- Every non-leaf node can host between *m* and *M* entries unless it is the root. Each entry is of the form (*MBR, P*), where *P* is a pointer to a child of the node and *MBR* is the smallest rectangle that spatially contains the MBRs contained in this child.
- The root node at least has 2 entries, unless it is a leaf.
- All leaves of the R-tree are at the same level.

Figure 2.6 (a) shows a set of 2-dimentional objects while Figure 2.6 (b) shows the corresponding R-tree index structure.

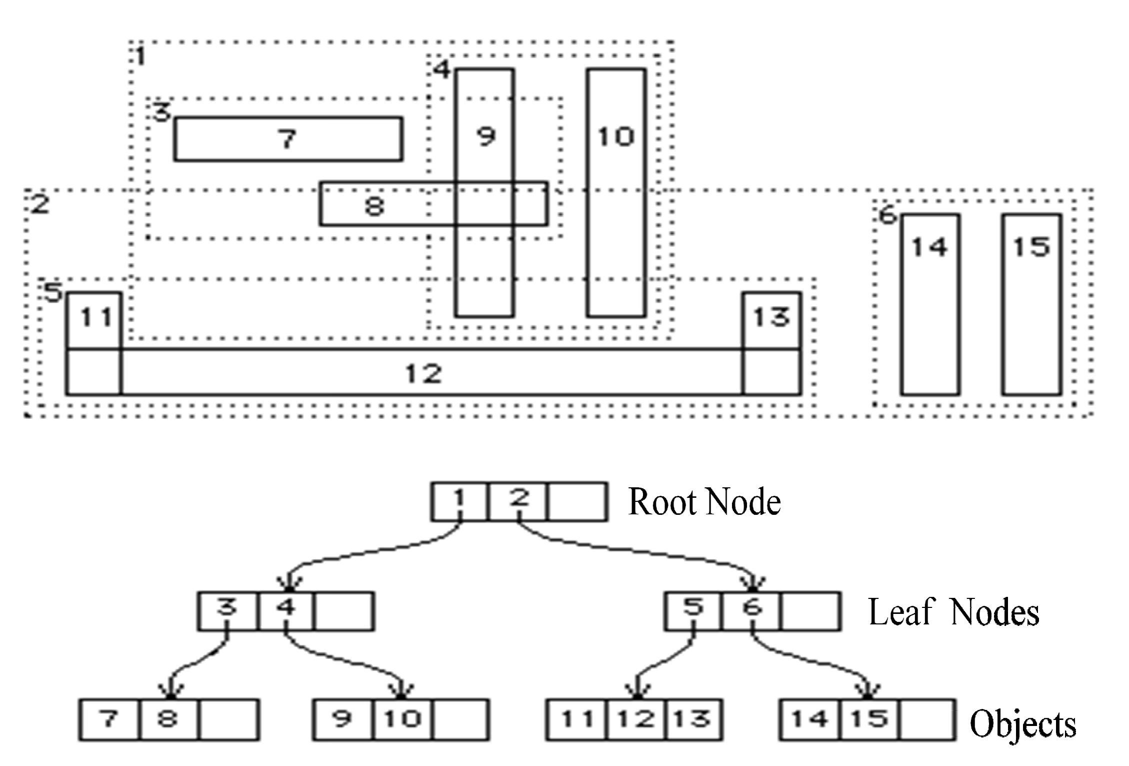


Figure 2.6: R-tree organization: (a) Spatial ordering (b) Tree representation.

R-trees are height-balanced dynamic data structures, i.e., global reorganization is not required to handle insertions or deletions. Insertions of new objects are directed to leaf nodes. At each level, the node that will be least enlarged is chosen. Thus, finally the object is inserted in an existing leaf if there is adequate space, otherwise a split takes place. Guttman (1984) proposed three alternative algorithms to handle splits, which are of linear, quadratic and exponential complexity; the criterion used is to minimize the two covering rectangles of the two resulting nodes.

2.2.3 R-tree Operations

- **Search:** Searching the R-tree is similar to searching B-tree. It descends the tree from the root, at each level selecting entries whose MBRs overlap with that of the query. When the leaf-node is reached, the entries whose MBRs overlaps with the MBR of the query object are selected. Because of the overlap between nodes, more than one sub tree may need to be visited.
- **Insertion:** New index records are inserted to the leaf nodes. Only one path of the tree is traversed, selecting an entry which requires the least enlargement to include the new object at each intermediate node until a leaf node is reached. If the selected leaf node overflows, the node is split. If needed, the splits and MBR adjustments propagate up the tree.
- **Deletion:** The Deletion algorithm searches the R-tree to locate the leaf node which contains the object to be deleted. The entry is removed from the leaf node. If node underflows, the entries of the node are re-inserted in neighboring nodes, then eliminating the node itself. An underflow occurs when the number of entries in a node becomes less than *m* after deleting an entry. MBR adjustment is repeated recursively on the parents of the node and continued till the root is reached.

- **Node Splitting**: Adding a new entry to a node which already has the full capacity (*M)* of entries will force the total (*M+1)* entries to be divided into two nodes. Division should have some characteristics that will ensure good overall performance of the index. It should be undertaken in a way that ensures the output nodes will unlikely be visited in a subsequent searches. Minimizing the area common to sibling nodes (overlap area) will minimize the number of nodes that must be visited by a search. Minimizing the total area spanned by the bounding rectangles of the sibling nodes (coverage area) reduces the possibility that sibling nodes are visited by the search. Minimizing the area in sibling nodes that is not occupied by the bounding rectangles of any of its children (dead area) ensures that objects that are spatially close to each other. These goals, at times, may be contradictory (Samet, 2004).

According To Guttman (1984), the splitting process is required to minimize the nodes coverage area. The example he gave for good splits and bad split is shown in figure 2.7. But his example is focused only on the coverage area minimization, while other factors are also important such as minimizing overlap between nodes.

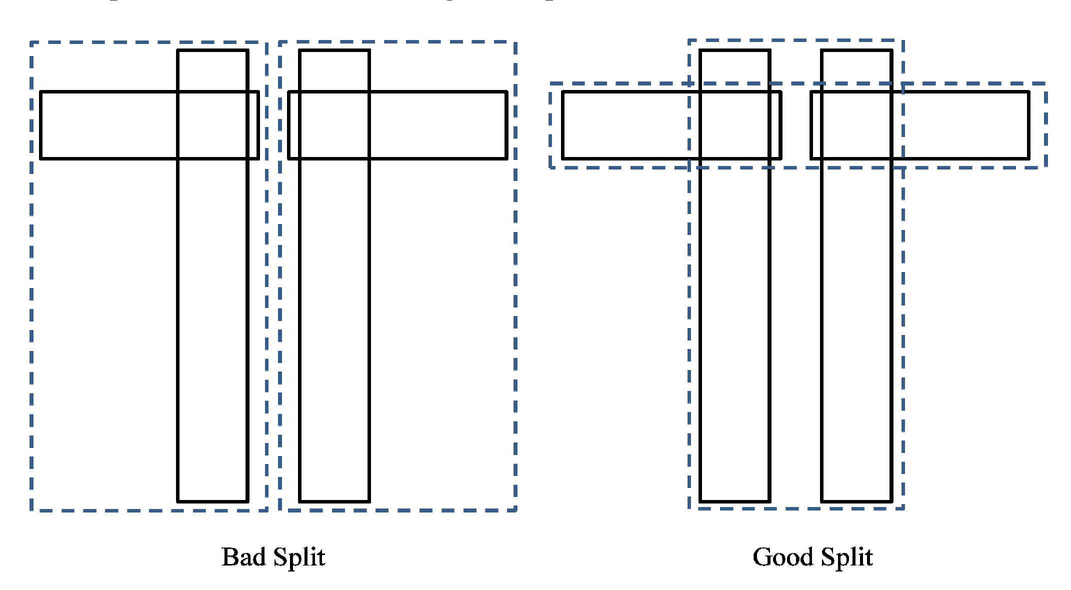


Figure 2.7: Guttmann's example of good split and bad split

Guttman (1984) proposed three splitting algorithms to split an over flown node. They differ mainly by its complexity; higher complexity produces better split:

- Linear-cost split: Choose two objects as seeds for the two nodes, where these objects are as furthest as possible. Then, consider each remaining object in a random order and assign it to the node requiring the smaller enlargement of its respective MBR.
- Quadratic-cost split: Choose two objects as seeds for the two nodes, where these objects if put together create as much empty space as possible (empty space is the space that remains from the MBR if the areas of the two objects are ignored). Then, until there are no remaining objects, choose for insertion the object for which the difference of empty space if assigned to each of the two nodes is maximized and insert it in the node that requires smaller enlargement of its respective MBR.
- Exponential-cost split: All possible groupings are exhaustively tested and the best is chosen with respect to the minimization of the MBR enlargement.

Guttman (1984) suggested using the quadratic algorithm as a good compromise between complexity and search efficiency.

2.3 R-tree variants: The original R-tree has two important disadvantages that motivated the study of more efficient variations: One is the possible performance deterioration due to the investigation of several paths from the root to the leaf level in executing a point location query, especially when the overlap of the MBRs is significant. The other is that a few large rectangles may increase the degree of overlap significantly, leading to performance degradation during range query execution, due to empty space.

In all R-tree variants, tree traversals for any kind of operations are executed in exactly the same way as in the original R-tree. Basically, the R-tree variations differ in the way they handle insertions and splits during insertions, by considering different

minimization criteria. These criteria, beside minimizing the sum of the coverage areas of the two resulting nodes, includes differing splits, minimizing the overlap area and reducing the time-cost of the original algorithms.

In the literature, there are tens of variations to the original R-tree. We will present some of the dynamic versions variation of the R-tree, in which objects are inserted on a one-by-one basis. As opposed to the static versions where a special packing technique can be applied to insert a priori known static set of objects into the structure, by optimizing the storage overhead and the retrieval performance (Manolopoulos et al., 2006).

2.3.1 R^+-Trees: the R^+-Trees (Sellis et al., 1987) was proposed as a structure that eliminates the multiple path visits in the R-tree for the point queries. To achieve this, R^+-Tree does not permit overlapping in the intermediate nodes of the tree.

The price of avoiding overlap between the MBRs of the non leaf nodes causes some of the inserted objects to be stored in more than one leaf node. This duplication will cause increasing tree height and more storage space. In spite of this disadvantage, the retrieval performance of the structure will improve since only a single tree path is going to be traversed.

Operation on R+-Trees is similar the R-Tree operations, except for the need to deal with the fact that multiple copies of an object's MBR may be stored in several leaf nodes. Insertion and deletion may require dealing with more than one node. In addition, an increased number of deletions may reduce storage utilization significantly. Therefore, appropriate reorganization must be performed to handle underutilized tree nodes. Splitting an over flown node is done by partitioning the node along on the main axes (vertical or horizontal), splits may require both upward and downward propagation of the tree.

2.3.2 R*-Tree: the R*-Tree (Beckmann, et al., 1990) was proposed as a structure that supports much more optimization criteria than just minimizing the nodes coverage area as in the original R-trees. These criteria include the minimization of MBR coverage area, the overlap area between the internal nodes' MBRs, the minimization of the MBR's margins (perimeters) to be more squared and the maximization of storage utilization of the leaf nodes. Some of these criteria may be contradictory to each other. For example, keeping the overlap and coverage area minimized will affect the storage utilization.

Insertion in R*-trees is an example of adopting more than one criterion. In order to insert a new object to the index, it selects the sub tree that will get the least increase in its MBR coverage area to receive the new object. However, in leaf nodes, the new object is inserted in the leaf node which its MBR will get the minimal overlap with neighboring nodes.

Splitting over flown nodes in R*-tree does not take place immediately. Instead, an attempt to reinsert some of the full leaf node's objects that are farthest from the center of that node. The reinsertion to the neighboring nodes hopefully will delay the need for splitting. The reinsertion also provides some kind of tree rebalancing and improves the query performance. Only one reinsertion operation is permitted for each level of the tree because it is a coasting operation.

If the overflow cannot be differed by the reinsertion operation, then node splitting is performed. Splitting algorithm has two steps; first, to pick a splitting axis, which is the axis with least perimeter among all dimensions, second, sort on the selected axis the entries of the over flown node by the lower then by the upper values of their rectangles, then choose the distribution with the minimum overlap value.

2.3.3 Hilbert R-tree: The Hilbert R-tree (Kamel and Faloutsos, 1994) was proposed as a structure that gives a deferred splitting approach in R-trees by ordering the R-tree nodes. The ordering approach groups similar data rectangles together, in order to minimize the area and perimeter of the resulting MBRs. The adopted ordering approach is the Hilbert space-filling curve. The Hilbert space-filling curve preserves the proximity of spatial objects well. Figure 2.3 (a) illustrated an example of a 4x4 Hilbert space-filling curve grid. The differed splitting means that instead of splitting one node into two nodes as in the original R-tree, the split will be differed until two nodes will be split into three nodes.

To insert an object in the Hilbert R-tree, the Hilbert value h of the center of the new MBR is used as a key. In each level, the node with minimum Largest Hilbert Value (*LHV*) of the siblings is chosen. When a leaf node is reached, the new object is inserted using its h value in the correct order. After insertion, the adjust tree algorithm is called to update the MBR and *LHV* value of the parent nodes. A split takes place only if all the node's siblings (s) are full and thus (s+1) nodes are produced.

When under flow occurs, the reverse is done by borrowing entries from the sibling nodes. If all siblings underflows, then a merging operation of ($s+1$) nodes into (s) nodes is done.

The Hilbert R-tree variant performance is vulnerable to large objects. Moreover, by increasing the space dimensionality, proximity is not preserved adequately by the Hilbert curve, leading to increased overlap of MBRs in internal tree nodes.

2.3.4 New Linear Node Splitting: Ang and Tan (1997) proposed a splitting algorithm called New linear to optimize the search performance by minimizing the overlap between the output nodes of the split process; less overlap will reduce the probability to search additional sub trees.

The criterion to avoid overlap in this algorithm during splitting process, is to push the MBRs of the objects as far apart as possible towards the boundary of the over flown node's MBR. This is done by calculating the distribution of the objects along all main axes; for each MBR, the distance of each border from the node's border along one of the main axes is recorded in a pair of lists for each axis, the pairs of lists are compared and the maximum value which represents the extreme distribution of the objects, determines the splitting axis, then splitting the node along that axis.

The New Linear algorithm does not apply to the minimum fill (*m*) constraint. Instead, if there is an output node with too less entries, it suggests reinserting these objects. Reinserted objects may be absorbed by the neighboring nodes, if not; they will reside in an underutilized node.

2.3.5 Optimal Node Splitting Algorithm: The algorithm proposed in (Garcia, et al., 1998) aims to find a polynomial time complexity for the Guttmann's optimal algorithm which has an exponential time complexity. The authors show that if the cost function used depends only on the characteristics of the MBRs, then the number of different MBR pairs is polynomial. Therefore, the number of different bipartitions that must be evaluated to minimize the cost function can be determined in polynomial time. The algorithm solves the optimal bipartition of $n=m+1$ entries, for d dimensions with running time of $O(n^d)$.

The proposed splitting algorithm investigates each of the $O(n^2)$ candidate pairs of MBRs, selects the one that minimizes the cost function, then assigns each rectangles to the corresponding MBR. Rectangles that are at the intersection of the best MBR pair are assigned following a selected criterion such as assigning them to the node with least number of entries.

2.3.6 Compact R-tree: The R-tree variant proposed by (Huang, et al., 2001), aimed to maximize the storage utilization in the original R-tree without influencing the index performance. It was motivated by the fact that the average storage utilization of the R-tree in the average is about 70% (Lomet, 1992). The Compact R-tree differs from the original version by the way it handles the insertions of a new entries into a full leaf node; if it is overflowed, one of its entries is selected to be moved to one of the sibling nodes which is best to accommodate that entry, then the new entry is inserted in the evacuated spot. Only if all leaf nodes are full, a split is done. The storage utilization of this variant reaches more than 97%, while the query performance is almost the same as the original index.

2.3.7 MBR Partition Policy Algorithm: In (Liu, et al., 2009), the authors proposed a new R-tree node splitting algorithm using MBR partition policy with linear time complexity. The algorithm partitions the over flown node according to its MBR shape and the shapes of the MBRs of its entries. For rectangles having the same area, the most square one will be with the smaller perimeter; a shape will be with smaller perimeter when the shape gets more squared. The algorithm tries to reduce the total coverage area of the output nodes by having their MBRs to be as square as possible.

The proposed partition first checks the shape of the over flown node's MDR, if it has length along one dimension more than the other dimensions by a predefined value, it will select to partition the node perpendicular to that dimension. Otherwise, it will check the shapes of the MBRs of all entries, to decide along which dimension the majority of objects have their length on, then partition the node along that dimension.

After the partition axis is selected, distribution of the entries is done by assigning each entry to the output node which totally contains that entry. For entries that are not totally

contained in any output node, distribution is done by selecting to add the entry to the node that will get the least enlargement, then the least overlap.

A balancing procedure is called after finishing distribution to verify that each node has the minimum fill constraint (*m*). If either of the output nodes received fewer entries than *m*, it will first try to repartition the node along another axis. In case this does not work, the closest entries to the under filled node are moved to it until *m* is reached.

Authors suggested a joint splitting with siblings when a square MBR, which most likely will not produce a square shape, is to be partitioned. The joint splitting idea is to join the over flown node with one of its siblings in order to do 2 to 3 splitting process instead of the regular 1 to 2 process.

2.3.8 Al-Badarneh, et al. Splitting Algorithm (Lowxy): In (Al-Badarneh, et al., 2010) the authors presented a new enhancement to the Guttmann's quadratic node splitting algorithm, the main goal of the proposed approach is to reduce the coverage area of the split process. Their enhancement consists of two parts; one part, an enhancement to the first step in the original Quad algorithm; PickSeeds algorithm. They proposed a linear cost pick seeds algorithm to replace of the original pick seeds algorithm which has a quadratic cost. Instead of exploring all the possibilities of picking two seeds that may cause the worst split, which will consume a quadratic time in number of entries by the original pick seeds algorithm; the proposed pick seeds algorithm suggests selecting the entry which its MBR corner has the lowest sum of the low *x* and *y* (*xl*, *yl*) values as the first seed and the entry which its MBR corner has the highest sum of the high *x* and *y* (*xh, yh*) values as the second seed. This operation of picking the seeds will take only a linear time to be done.

The other part of their enhancement is to solve the remaining entries problem in the original distribute algorithm. Remaining Entries problem that the quadratic split algorithm of the original R-tree suffers from is; while distributing the $(M+1)$ entries, if one of the new nodes has reached the maximum number of entries $(M - m + 1)$, all remaining entries are going to be assigned to the other node in order to reach the minimum fill (m) value. This is done without considering the geometric features of the remaining entries and this may cause a very non-optimized splitting of that particular node. The result is a split with uneven distribution of the entries which reduces the storage utilization and increases the coverage area.

The enhancement to the original distribute algorithm suggests that when distributing the entries between the output nodes, all $(M+1)$ entries of the over flown node are sorted twice; first sorting the entries upper corners distance from the first seed's lower corner, then sorting the entries lower corner distance from the second seed's upper corner. Distribution of the entries is done by assigning to the first output node the nearest m entries to seed1 and to second output node the nearest m entries to seed2. Remaining entries $(M-2m+1)$ are assigned one by one to the nearest node according to the same criterion. The time complexity of the proposed algorithm is $(nlogn)$.

2.3.9 Wang, et al. Splitting algorithm: In (Wang, et al., 2010) authors proposed a new enhancement to the original Quad Algorithm by proposing a new pick seeds and distribute procedures. The proposed algorithm considers reducing overlap area between output nodes first and to decrease the coverage area in the second place. Multiple path queries could be avoided or decreased and a more effective R Tree is constructed.

The new algorithm first measures the distance between each entry to the other entries in the over flown node, keeps the data in an ordered matrix, then uses the data to pick

seeds and distribute the entries. The adopted criterion to measure the distance between the object in a node is depicted in figure 2.8.

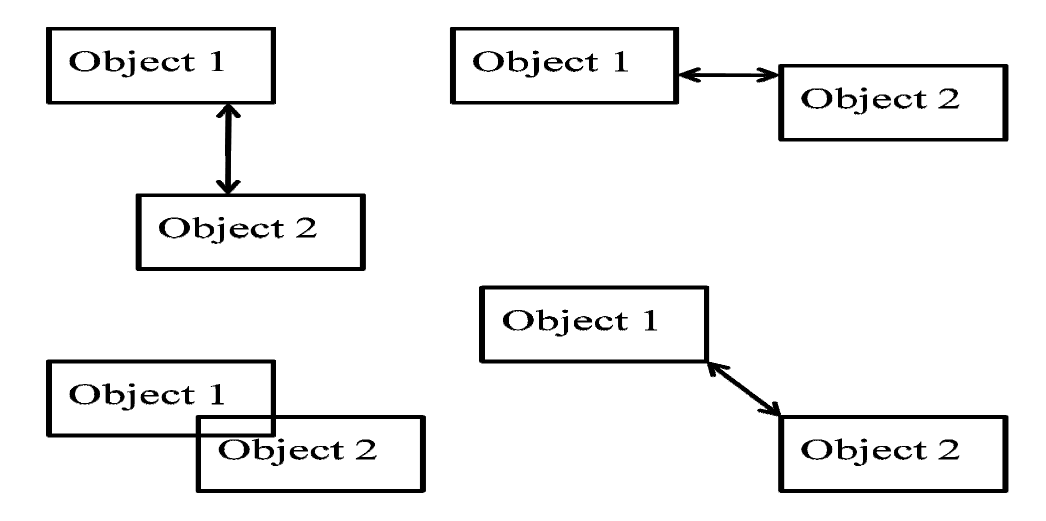


Figure 2.8: Measuring objects distance criterion used by Wang, et al.

The ordered matrix's highest value (i.e. the two entries that have the maximum distance between each other) is selected as the seeds for the split. After the seeds are selected, the next entry in the ordered matrix is assigned to the output node with the nearest seed, until all entries are assigned. If either of the output nodes reaches the maximum fill (*M*-*m*), all remaining entries are moved to the other node.

CHAPTER 3

Proposed Splitting Method (CBS Algorithm)

In this chapter, we introduce a new technique to split an R-tree over flown node, named the Corner Based Splitting algorithm (CBS). The CBS algorithm is easy to implement and has better results in reducing the total output nodes' coverage area while minimizing the overlap area. The CBS algorithm shows a better overall performance when compared with two other splitting algorithms, namely the Quad algorithm and the algorithm proposed in (Al-Badarneh, et al., 2010), which we will refer to it as the Lowxy algorithm (Yaseen, 2006).

3.1 Node Splitting

Splitting the covering rectangle of an over flown node N will produce two rectangles describing the covering areas of the two output nodes N1 and N2. The splitting process should try to minimize the total coverage and overlap areas of the output rectangles. In order to minimize the coverage area and the overlap area, the geometrical properties of rectangles necessitate that the splitting process should better avoid a diagonal split, it should rather split along one of the data axes, which in a case of two dimensional spatial data is either the x-axis or the y-axis.

To demonstrate the importance of avoiding a diagonal split, let's consider the example in Figure 3.1. Different possible splits of an over flown node N are given. It is clear that splitting a rectangle diagonally produces the most total coverage area and the most overlap amount between the two newly created rectangles, while splitting along one of the main axes produces the least total coverage area and overlap between the output nodes N1 and N2.

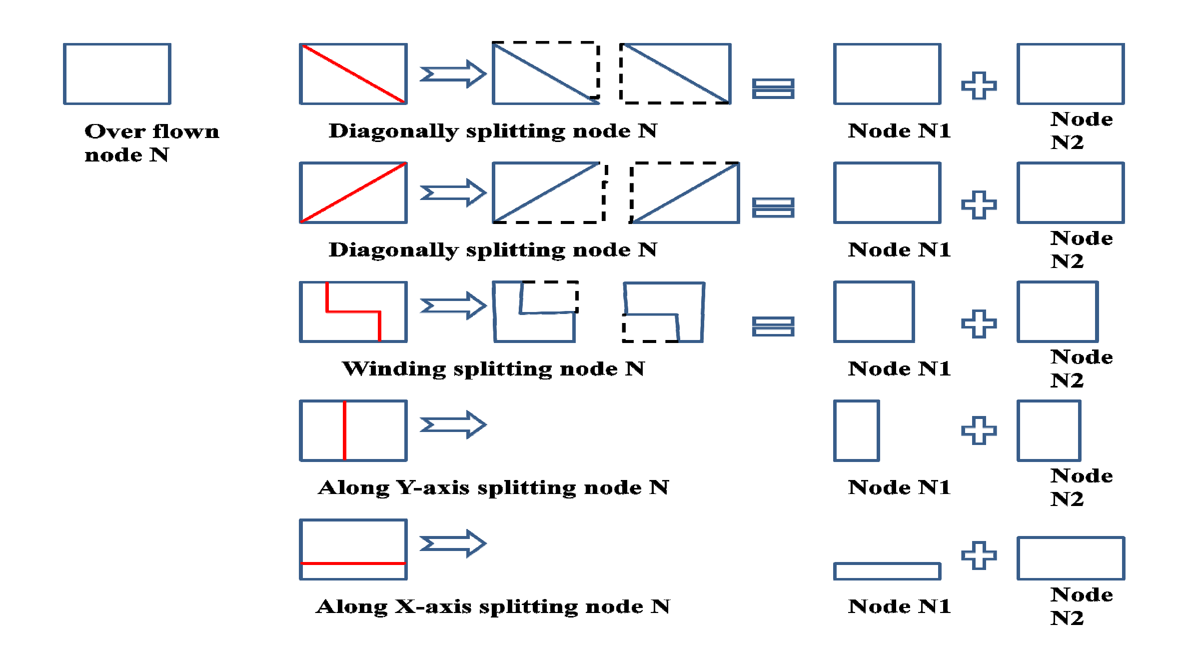


Figure 3.1: Different possible splits of an over flown node N

The proposed algorithm takes advantage of the above discussion, in order to split an R-tree over flown node, the CBS algorithm works by splitting the node along one of its main axes. Although the CBS algorithm can deal with multi-dimensional spatial data nodes, we will present here the algorithm's work mechanism for the case where only two dimensional spatial data is used in order to simplify the discussion.

An MBR is specified by its lower and upper corners, lower corner specifies the lowest x-axis and y-axis values (X_l, Y_l) and upper corner specifies the highest x-axis and y-axis values (X_h, Y_h). The other two corners of the bounding rectangle are specified by (X_l, Y_h) and (X_h, Y_l) respectively. We will refer to these four corners as C_0: *(X_l, Y_l)*, C_1: *(X_h, Y_l)*, C_2: *(X_h, Y_h)*, C_3 *(X_l, Y_h)*. Figure 3.2 illustrates the naming convention.

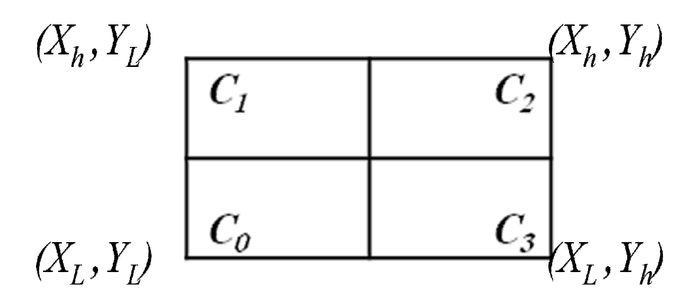


Figure 3.2: MBR's Corners naming convention.

To decide along which axis an over flown node should be split into two new nodes; we define a counter for each corner, calculate the center of each entry (object) in node N, record the object with the nearest corner by incrementing that corner's counter.

By the end of previous step, we have the count of near objects for every corner. Next, we combine objects belonging to two corners to form the first output node N1 and objects belonging to the other two corners to form the second output node N2. Using the fact that diagonally faced corners must not be joined in one node; the probabilities left to form one of the two output node are either to join C_0 with C_1, or to join C_0 with C_3. If we join C_0 with C_1 then we are splitting node N along the y-axis, as well as joining C_0 with C_3 is splitting node N along the x-axis. Objects belonging to the other two corners are combined to form the second output node N2.

The decision on which axis to split the over flown node on is taken upon the corners' counters, assuming that C_0 will reside in N1 and C_2 will reside in N2, we compare the values of C_0 counter and C_2 counter. If C_0 counter is greater than C_2 counter, among the remaining corners (C_1, C_3), we join with C_0 the corner which has the least counter value (either C_1 or C_3), if not, we join with C_0 the corner which has the most counter value. As a result of choosing which corners to be joined, the splitting axis is decided and the entries are distributed evenly as possible between the two new nodes.

In the case where either one of the output nodes will receive fewer entries than the minimum fill value allowed, we check the entries of the full node for the nearest object's center to the splitting axis, that object is moved to the node in need. This process is repeated until the minimum number of entries is fulfilled.

3.2 The CBS algorithm pseudo code

the pseudo code for the CBS algorithm to split an over flown node *"N"* with *"M+1"* entries represented by its minimum bounding rectangle MBR $((X_l, Y_l), (X_h\ Y_h))$ which

has 4 corners; C_0 (X_l, Y_l), C_1 (X_h, Y_l), C_2 (X_h, Y_h), C_3 (X_l, Y_h). Node N should be split into two new nodes N_1 and N_2.

1. Find the Covering Rectangle center using the formula:

   $(\text{CovRect}X_{\text{cen}}, \text{CovRect}Y_{\text{cen}}) = (((X_h + X_l)/2), ((Y_h + Y_l)/2))$

2. For each object rectangle $((x_{Min}, y_{Min}), (x_{Max}, y_{Max}))$ in node N,

   2.1 Calculate each object center using formula:

   $(\text{Obj}X_{\text{cen}}, \text{Obj}Y_{\text{cen}}) = ((x_{Max} + x_{Min})/2), ((y_{Max} + y_{Min})/2)$.

   2.2 Classify Object into one of the four Corner' groups:

```
If ObjXcen > CovRectXcen
    If ObjYcen > CovRectYcen
        Assign Object to Corner 2 (C2 Group), increment C2 counter.
    Else
        Assign Object to Corner 3 (C3 Group), increment C3 counter.
    Endif
Else
    If ObjYcen > CovRectYcen
        Assign Object to Corner 1 (C1 Group), increment C1 counter.
    Else
        Assign Object to Corner 0 (C0 Group), increment C0 counter.
    Endif
Endif
```

3. Distribute the entries according to the corners' counters:

```
3.1     If Count (C0) > Count (C2)
            Move objects of C0 to N1, Move objects of C2 to N2
        Else
            Move objects of C2 to N1, Move objects of C0 to N2
        Endif
3.2     If Count (C1) > Count (C3)
            Move objects of C1 to N2, Move objects of C3 to N1.
        Else
            If Count (C3) > Count (C1)
                Move objects of C3 to N2, Move objects of C1 to N1.
            Else    /* Tie break */
                Choose to move objects of C1 OR objects of C3 to N1
                according to the least overlap.

                If still Tie
                    Choose to move objects of C1 OR objects of C3 to
                    N1 according to the least total coverage area.
                Endif
            Endif
        Endif
```

4. In case either one of the output nodes received less objects than the minimum fill value allowed, check the entries of the full node for the nearest object's center to the splitting axis, move that object to the node in need, repeat until the minimum fill value is fulfilled.

To demonstrate the CBS algorithm's work, a step by step example using drawings is illustrated in Figure 3.3.

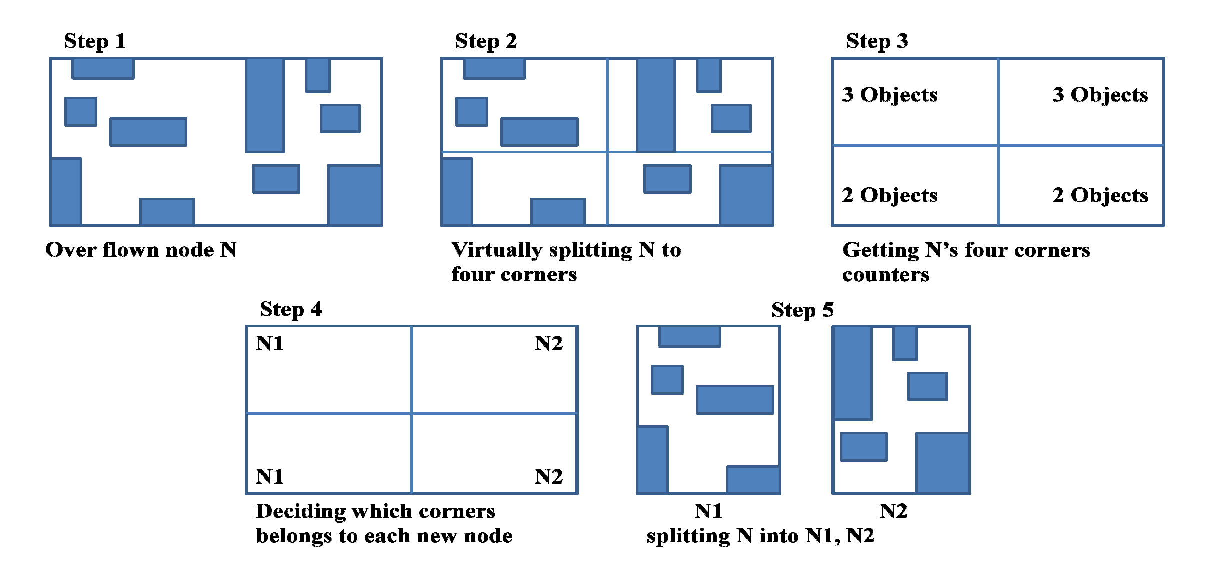


Figure 3.3: The CBS algorithm's work, a step by step example

Step 1: Shows an R-tree over flown node N with 10 entries.

Step 2: The center of both x and y axes of the over flown node covering rectangle is calculated to get the four corners of N, the center point of the rectangle is given by the formula $(((X_h + X_l) /2), ((Y_h + Y_l) /2))$.

Step 3: For each corner, count the entries which are nearest to that corner, to do so; we use the centers of objects which are calculated using the same formula in step 2. By testing each object's center x-value against the covering rectangle center x-value, it tells if that object belongs to the upper or lower corners, then by testing the object's center y-value against the covering rectangle center y-value, it tells if that object belongs to left or right corner. When the nearest corner for an object is decided, the object is registered with that corner. Repeating step 3 for all entries provide us with the total number of objects for each corner.

Step 4: The counters of the corners decide which axis to split node N on, this is done by comparing counter values of diagonally faced corners. First comparing counter values of corner C_0 and corner C_2, the largest one is assigned to N1, the other is assigned to

N2. Same is done for the counters of corner C_1 and C_3, but with reverse comparison, where the least is joined to N1, the other is joined to N2.

Step 5 shows the final output of the CBS algorithm.

3.3 Graphical comparison

A quick graphical comparison between different splitting methods is given in Figure 3.4. Over flown node N with 10 entries is shown in Figure 3.4 (a). Output nodes must have at least 5 entries when the minimum fill is set to 0.5 and 3 entries when minimum fill is set to 0.33.

Splitting node N using the Quad algorithm with minimum fill 0.5 is shown in Figure 3.4 (b), the output nodes total coverage ratio is 116% and overlap ratio is 25%. Splitting node N using the Quad algorithm with minimum fill 0.33 is shown in Figure 3.4 (c), the output nodes total coverage ratio is 89% and overlap ratio is 3%.

Splitting node N using the Lowxy algorithm with minimum fill 0.5 is shown in Figure 3.4 (d), the output nodes total coverage ratio is 125% and overlap ratio is 31%. Splitting node N using the Lowxy algorithm with minimum fill 0.33 is shown in Figure 3.4 (e), the output nodes total coverage ratio is 126% and overlap ratio is 26%.

Splitting node N using the CBS algorithm with minimum fill 0.5 is shown in Figure 3.4 (f), the output nodes total coverage ratio is 84% and overlap ratio is 0%. Changing the minimum fill value does not change the shape of output nodes.

Splitting node N using the New Linear algorithm is shown in Figure 3.4 (f), the output nodes total coverage ratio is 96% and overlap ratio is 6%. New Linear algorithm doesn't adapt the minimum fill constraint.



Figure 3.4: Graphical comparison between different splitting algorithms.

Comparison between the size of input node (N) and the output nodes (N1, N2) of the different algorithms in Figure 3.4 shows that the CBS algorithm has the least ratio of coverage area of all. In addition, all algorithms produced some overlap between the output nodes except for the CBS algorithm which has a zero overlap ratio. Table 3.1 lists the coverage area ratios and the overlap ratios of Figure 3.4 for each algorithm.

Table 3.1: Coverage area ratio and overlap ratio of Figure 3.4 for different algorithms

	Coverage Ratio	Overlap Ratio
Quad alg., m= 0.5	116 %	25 %
Quad alg., m= 0.33	89 %	3 %
Lowxy alg., m= 0.5	125 %	31 %
Lowxy alg., m= 0.33	126 %	26 %
CBS alg.	84 %	0 %
New Linear alg.	96 %	6 %

Chapter 4
Experiments and Results

This chapter lists the performance experiments done to explore the properties of the CBS algorithm and the performance comparison tests to compare the CBS algorithm against two other R-tree's node splitting algorithms.

The selected R-tree node splitting algorithms were: First is the original R-tree's Quadratic algorithm, because it is the most node splitting algorithm being used to test against by newly suggested node splitting methods. Second is the algorithm proposed in (Al-Babarneh, et al.,2010), because it is relatively close to the CBS algorithm since it works by selecting the lowest and highest corners to be the seeds for the node splitting process. We will refer to this algorithm as the Lowxy algorithm.

All algorithms implementations used in the experiments were written in C++ language and were run at Pentium M 1.4 GHz machine with 1 GB of RAM running Windows XP.

4.1 Test Data

The data files used to test the three algorithms were both synthetic and real world data files. The synthetic data files were generated using a C++ program and the real world data files were obtained from the internet.

4.1.1 Synthetic data: Both Uniform distribution and Normal (Gaussian) distribution types were considered in running the experiments. A data file for each distribution type was randomly generated using C++ generator, each file consists of 100000 entries (MBRs), the maximum rectangle area size was set to be 0.001 of the world space area size. We will refer to these files as the Uniform data file and the Normal data file.

4.1.2 Real world data: Popular Spatial (geographical) Datasets in 2D space test data files were used to run the experiments, we used the following files from the Tiger/Line Geography Division, U.S. Census Bureau (www.census.gov/geo/www/tiger) obtained from the rtreeportal web site (www.rtreeportal.org):

- **California stream file**: Containing the MBRs of 98451 streams (poly lines) of California. We will refer to this file as the CA file.
- **Long Beach stream file**: Containing the MBRs of 53145 of Long Beach county roads. We will refer to this file as the LB file.
- **North East data set**: Containing 123593 postal addresses (points), which represent three metropolitan areas (New York, Philadelphia and Boston). We will refer to this file as the NE file.

4.2 Experiment Settings

There are important settings in running the tests of node splitting; the maximum fill value (Max fill) and the minimum fill value (Min fill) of a node. Max fill value is set according to the disk page size to be used or tested. Since a disk page has a specific amount of bytes; a single page can hold up to a specific number of entries (MBRs).

An entry (a two dimensional rectangle) requires four bytes for each of the four numbers describing the lower left and upper right corners of its MBR, with additional four bytes for its pointer which point to the object, in total, each entry (MBR) requires twenty bytes to be represented (Guttman, 1984). Table 4.1 lists examples of the maximum number of entries (MBRs) that can fit in a single disk page for some disk page sizes.

Table 4.1: Maximum node entries for selected disk page size values

Page Size	MAX MBR Number = INT(Page size/20)
0.5KB	25
1KB	50
2KB	102
4KB	204

The minimum fill (Min fill) value of a node resembles the least number of entries a node can have as a result of the split process. Table 4.2 presents some of the possible minimum fill values that can be chosen to test with according to the Max fill value.

Table 4.2: Minimum fill values according to the Max fill values.

MAX	INT (MAX/2)	INT (MAX/3)	INT (MAX/4)	INT (MAX/5)	INT (MAX/6)	INT (MAX/7)
25	12	8	6	5	4	3
50	25	16	12	10	8	7
102	51	34	25	20	17	14

We have conducted several experiments to test the CBS algorithm against the other two algorithms, these experiments can be grouped into two major types:

- Index creation tests: Where we compared the quality of the index created using each of the three algorithms, we compared the index creation time, the number of created nodes and the quality of the splitting process of the over flown nodes while index being created by each algorithm. Different minimum fill values with different types of test files were used.
- Index query tests: Where we compare the intersection query performance among the three algorithms by recording the needed I/O operations to fulfill each query by each of the three algorithms. Different sizes of queries with different types of test files were used.

4.3 Index creation tests

During index creation, the node splitting process is called only after many insertions when a node overflows. But the splitting process plays a key factor of deciding the final shape of the tree; this final shape is responsible of the index performance.

Creation time duration and the number of created index nodes are used here as metrics to compare the three algorithms effectiveness. Other metrics can be used to measure the quality of the splitting process.

4.3.1 Index creation time comparison: We compared the time taken by each of the three algorithms to create an index using the Uniform data file with 100000 entries when Max fill = 50 and Min fill = Max/2. The ratio of time taken by Quad Algorithm to the time taken by CBS algorithm is 1.11% and the ratio of time taken by Lowxy Algorithm to the time taken by CBS algorithm is 1.55%. Results show that the CBS algorithm took the least time to create the index, with the ratio of 11% time saving than the Quad algorithm and 55% time saving than the Lowxy algorithm.

4.3.2 Number of index nodes comparison: The number of index nodes when created by each of the three algorithms using the Uniform data file when Max fill = 50, Min fill = Max/2, for different number of entries are presented in table 4.3. While table 4.4 presents the number of index nodes when created by each of the three algorithms using Uniform data file when Max fill = 50, 100000 entries, for different values of Min fill.

Table 4.3: Number of Index nodes created using Uniform data file with Max=50, Min = Max/2 for different number of entries.

	No. of Entries / No. of Index Nodes	25000	50000	75000	100000
Quad Alg.	Internal Nodes	21	43	63	85
	Leaf Nodes	711	1439	2171	2880
	Total Nodes	732	1482	2234	2965
Lowxy Alg.	Internal Nodes	22	44	69	94
	Leaf Nodes	729	1467	2221	2956
	Total Nodes	751	1511	2290	3050
CBS Alg.	Internal Nodes	23	42	69	84
	Leaf Nodes	707	1427	2153	2869
	Total Nodes	730	1469	2222	2953

Table 4.4:.Number of Index nodes created using Uniform data file with Max=50, 100000 entries for different Min fill values.

	Min fill / No. of Index Nodes	Max/2	Max/3	Max/5	Max/7	Max/9
Quad Alg.	Internal Nodes	85	94	118	178	319
	Leaf Nodes	2880	3029	3695	4507	5925
	Total Nodes	2965	3123	3813	4685	6244
Lowxy Alg.	Internal Nodes	94	92	105	99	95
	Leaf Nodes	2956	2954	3017	3016	3014
	Total Nodes	3050	3046	3122	3115	3109
CBS Alg.	Internal Nodes	84	90	88	88	88
	Leaf Nodes	2869	2923	2929	2929	2929
	Total Nodes	2953	3013	3017	3017	3017

Comparing the total number of created nodes in table 4.3 and table 4.4 for the three algorithms shows that the CBS algorithm always has the least number of nodes, even when changing the number of entries or the minimum fill value.

Figure 4.1 presents a comparison of total created index nodes number between the three algorithms using Uniform data file, Max fill = 50, Min fill = Max/2 and 100000 entries.

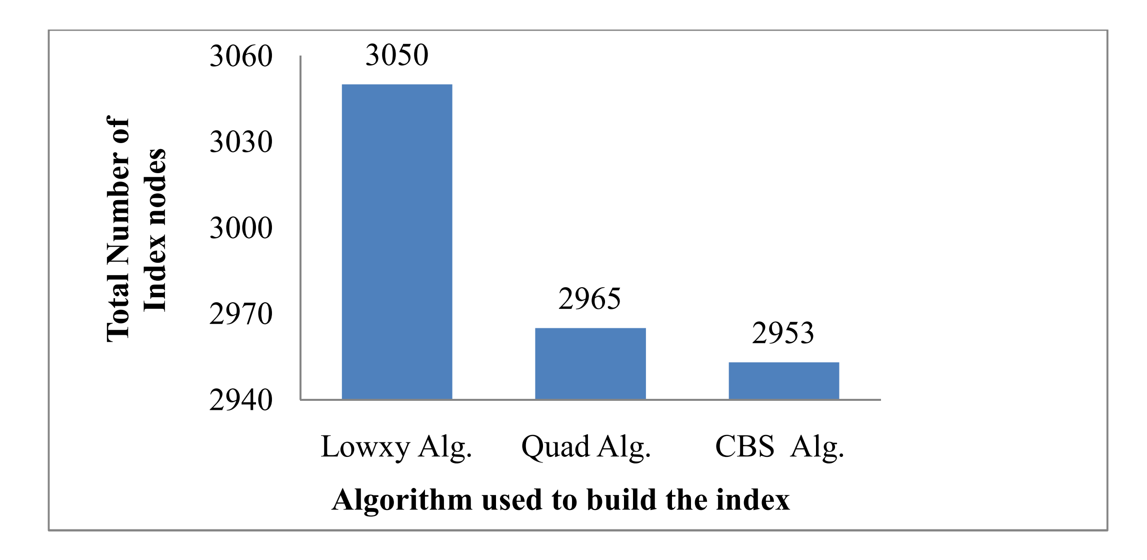


Figure 4.1: Comparison of total index nodes created by each algorithm when Max=50, Min=Max/2, for 100000 entries using Uniform data file

Figure 4.2 presents a comparison of the total created index nodes number between the three algorithms when using Uniform data file with Max fill = 50, 100000 entries, for different Min fill values.

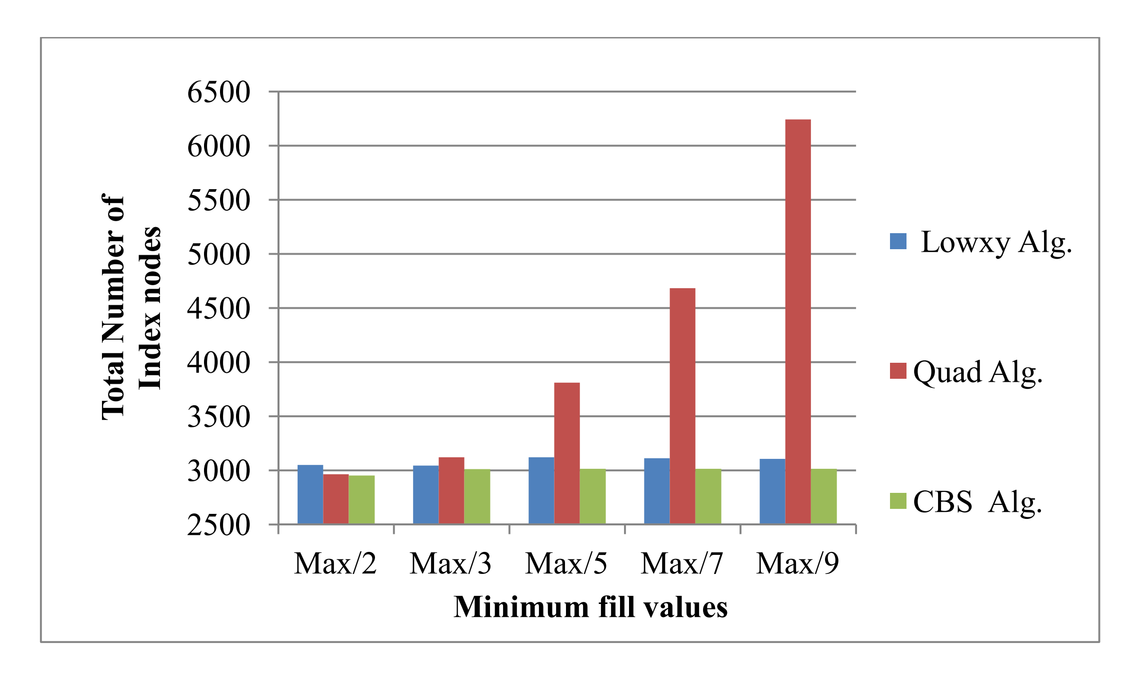


Figure 4.2: Comparison of total index nodes created by each algorithm when Max=50, 100000 entries, for different Min fill values using Uniform data file.

An important observation from table 4.4 about the CBS algorithm, that beside being always producing the least number of nodes between the three algorithms, it also produces the same number of nodes when decreasing the Min fill value to be less than Max/3; this implies that the index structure and the time taken to build the index is the same for these values and that the CBS algorithm distributes the entries evenly between output nodes. Same does not apply for the other two algorithms. Figures 4.3, 4.4 and 4.5 presents the number of total created nodes for each of the Lowxy, Quad, and CBS algorithms respectively.

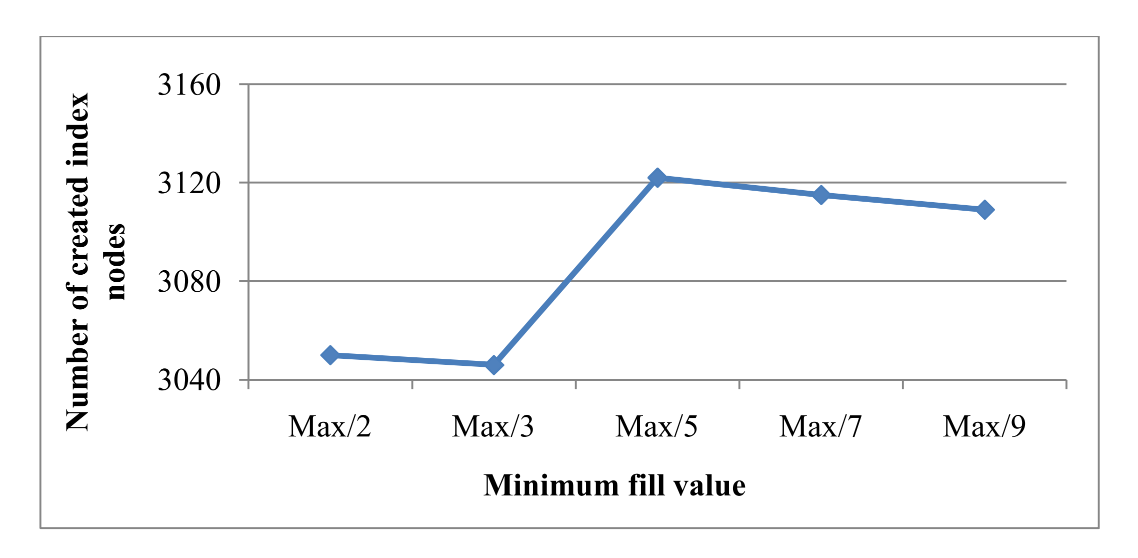


Figure 4.3: Number of created index nodes by the Lowxy alg. using the Uniform data file with 100000 entries.

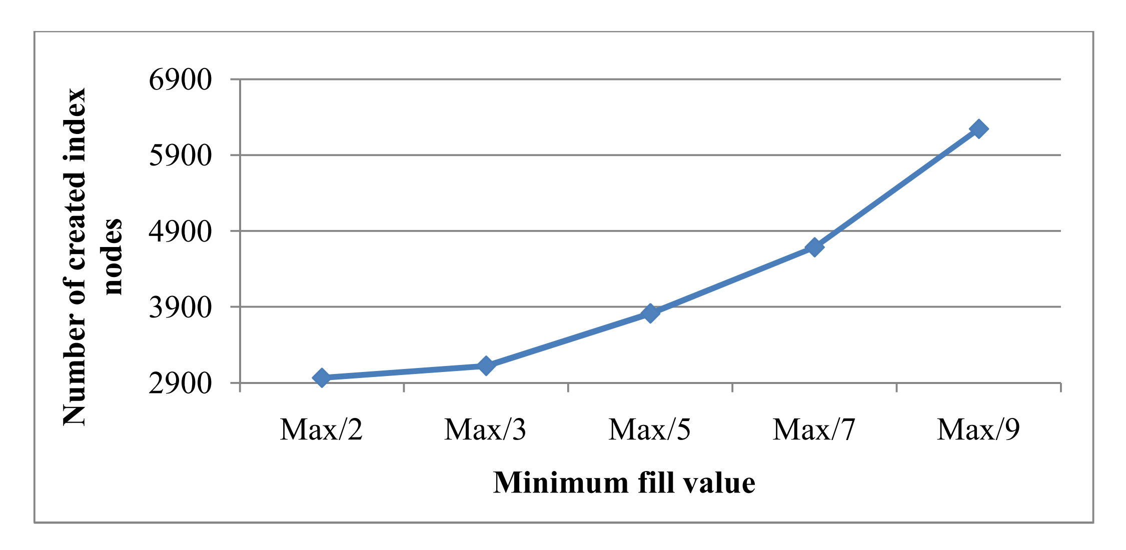


Figure 4.4: Number of created index nodes by the Quad alg. using the Uniform data file with 100000.

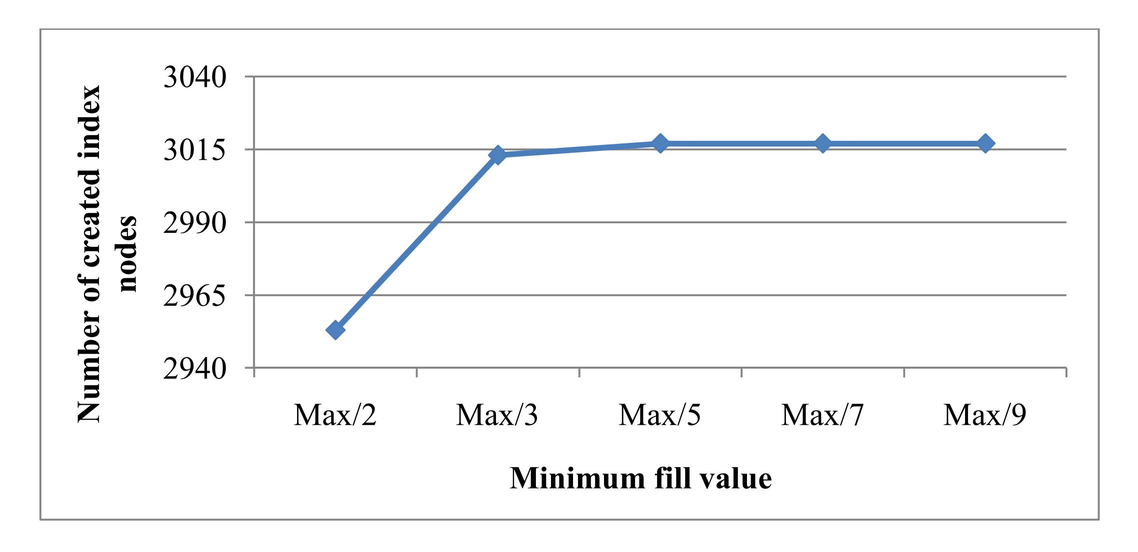


Figure 4.5: Number of created index nodes by the CBS alg. using the Uniform data file with 100000 entries.

When using the Normal data file to build the indexes by the three algorithms, a slit difference to the number of created nodes occurs; the CBS algorithm keeps having the least number of created nodes except for the case where Min fill is Max/2, in which the CBS algorithm comes in the second place after the Quad algorithm. This change can be attributed to the characteristics of the normal distribution, where high concentration of entries is in a relatively small area.

Table 4.5 presents the number of index nodes when created by each of the three algorithms using the Normal data file when Max fill = 50, 100000 entries for different values of Min fill. While Figure 4.6 presents a comparison of the total created index nodes number between the three algorithms shown in table 4.5.

Table 4.5:.Number of Index nodes created using Normal data file with Max=50, 100000 entries, for different Min fill values.

	Nodes	Max/2	Max/3	Max/5	Max/7	Max/9
Quad	internal	83	91	129	169	235
	leaf	2843	2978	3572	4258	5381
	total	2926	3069	3701	4427	5616
Lowxy	internal	85	90	94	97	102
	leaf	2902	2978	2945	2978	2989
	total	2987	3068	3039	3075	3091
CBS	internal	90	93	90	95	91
	leaf	2878	2908	2910	2902	2916
	total	2968	3001	3000	2997	3007

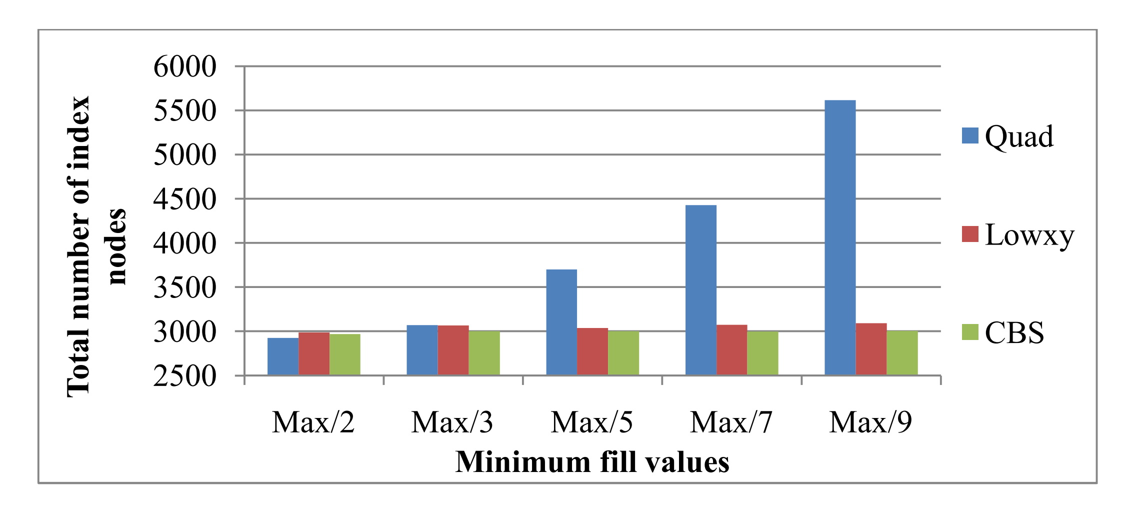


Figure 4.6: Comparison of total index nodes when max=50, 100000 entries, for different Min fill values when using Normal data file

4.3.3 Splitting quality comparison: The metric we use here to measure the splitting quality of the over flown leaf nodes is the amount of area overlap between the two newly created nodes (i.e. area common to both nodes). Here we will refer to the over flown leaf node as N, the output leaf nodes as N1 and N2. The overlap ratio is calculated by accumulating the area which is common to the both covering MBRs of N1 and N2 in each leaf node split, then dividing the accumulated area by the total number of leaf nodes. Table 4.6 and figure 4.7 presents the abovementioned metric for indexes when created by each of the three algorithms using the Uniform data file when Max fill = 50, 100000 entries, for different Min fill values.

Table 4.6: Overlap ratio between the output nodes of the split process using Uniform data file with 100000 entries, Max=50 and different Min fill values.

Min fill Value	Lowxy Alg.	Quad Alg.	CBS Alg.
Max/2	22.93%	10.14%	5.41%
Max/3	20.18%	8.15%	4.51%
Max/4	19.92%	6.72%	4.59%
Max/5	19.20%	6.05%	4.59%

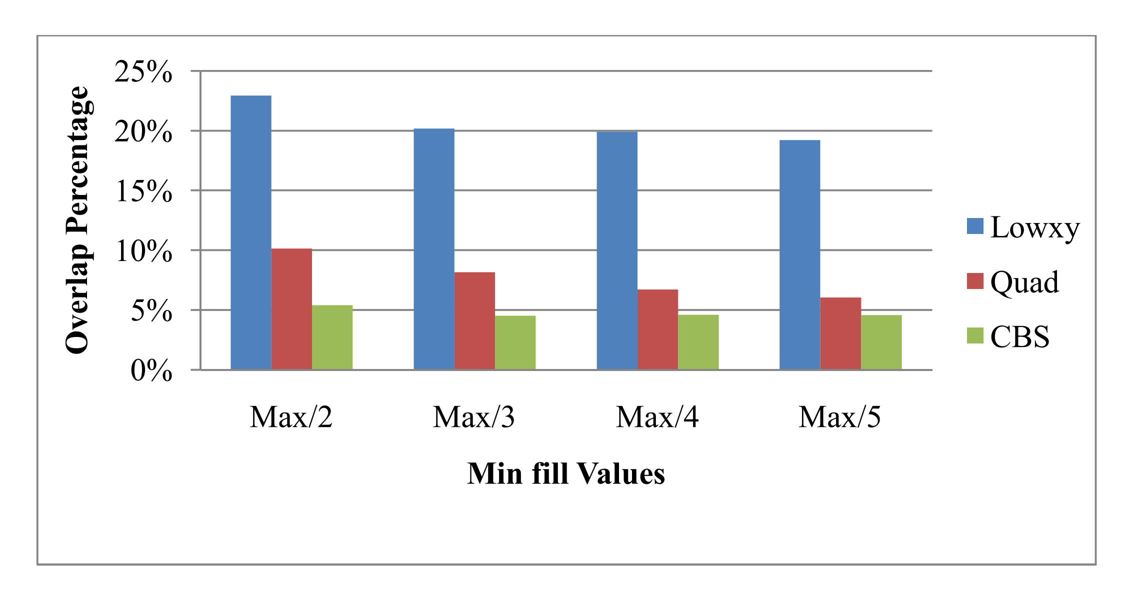


Figure 4.7: Overlap percentage between output nodes of the split process for the three algorithms using Uniform data file with 100.000 entries, Max=50, with different Min fill values.

Table 4.6 and Figure 4.7, shows that the CBS algorithm produces the least overlap ratio among the three algorithms, when Min fill = Max/2 the CBS algorithm overlap ratio is less than the Quad algorithm ratio by 47% and less than the Lowxy algorithm by 76%. When Min fill value is decreased, the ratio decreases between CBS and Quad algorithms, but it is almost steady with the Lowxy algorithm. When Min fill = Max/5 the overlap ratio of the CBS algorithm is less than the Quad algorithm by 24% and less than the Lowxy algorithm by 76%.

The overlap test results when using the three algorithms to create indexes for the NE file, which contains 123593 entries of postal addresses (points), shows that the overlap ratio for the CBS algorithm was 0%. This result shows that using point data or entries with very small areas will get the CBS algorithm to produce almost no overlap at all, which is not the case for the other two algorithms. Figure 4.8 presents these results.

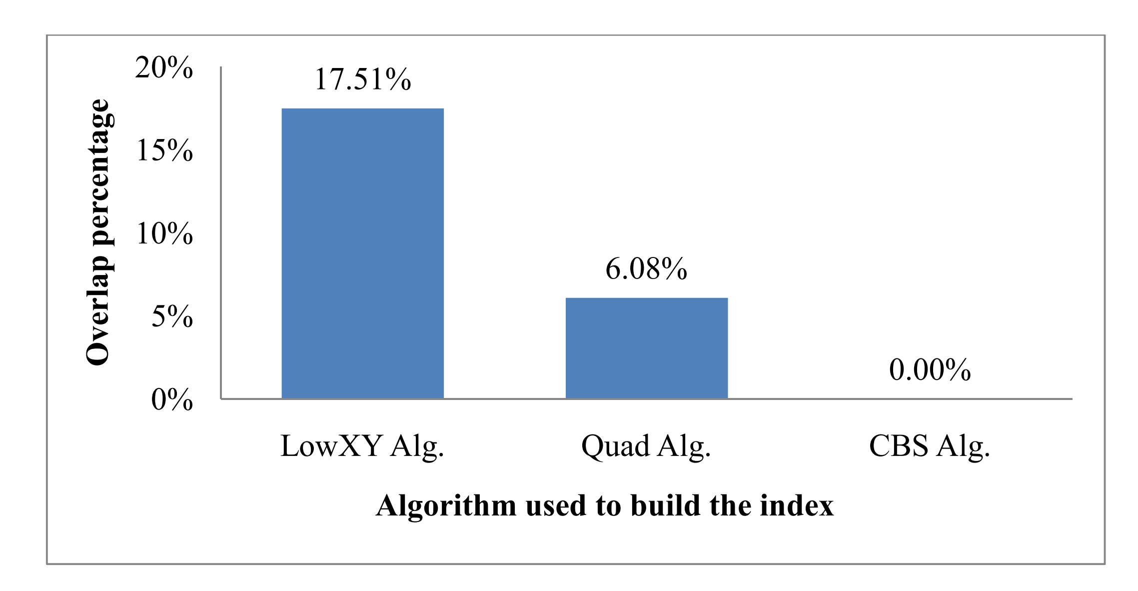


Figure 4.8: Overlap Ratio for the NE data file with 123593 entries (points), Max= 50 and Min fill = Max/2.

Although Uniform and Real data files overlap tests gives the CBS algorithm the advantage over the other two algorithms, this is not the case when using the Normal data file. The overlap results for creating indexes by the three algorithms using Normal data file with 100000 entries, Max=50 and different Min fill values are presented in

table 4.7. The overlap ratio when Min Fill value is Max/2 and Max/3 for the Quad algorithm and the CBS algorithm is almost the same. However, when the Min fill value is decreased, the overlap ratio of the Quad algorithm becomes less than that of the CBS algorithm. The overlap ratio of the CBS algorithm when Min fill = Max/5 is 12% more than that of the Quad algorithm. This change in the CBS algorithm behavior can also be attributed to the nature of the normal distribution of entries, where the entries are condensed in a relatively small portion of the world space.

Table 4.7: Overlap ratio between output nodes of the split process using Normal data file with 100000 entries, Max=50, with different Min fill values

Min fill Value	Lowxy Alg.	Quad Alg.	CBS Alg.
Max/2	31.29%	26.12%	26.08%
Max/3	30.17%	25.64%	25.78%
Max/4	29.93%	24.11%	25.56%
Max/5	29.45%	22.87%	25.55%

Compared with Lowxy algorithm, the CBS algorithm overlap ratio is less for all Min fill values by 20% to 15%. Figure 4.9 presents the comparison of overlap ratio between the three algorithms when creating indexes using Normal data file, with 100000 entries, Max = 50, for different Min fill values.

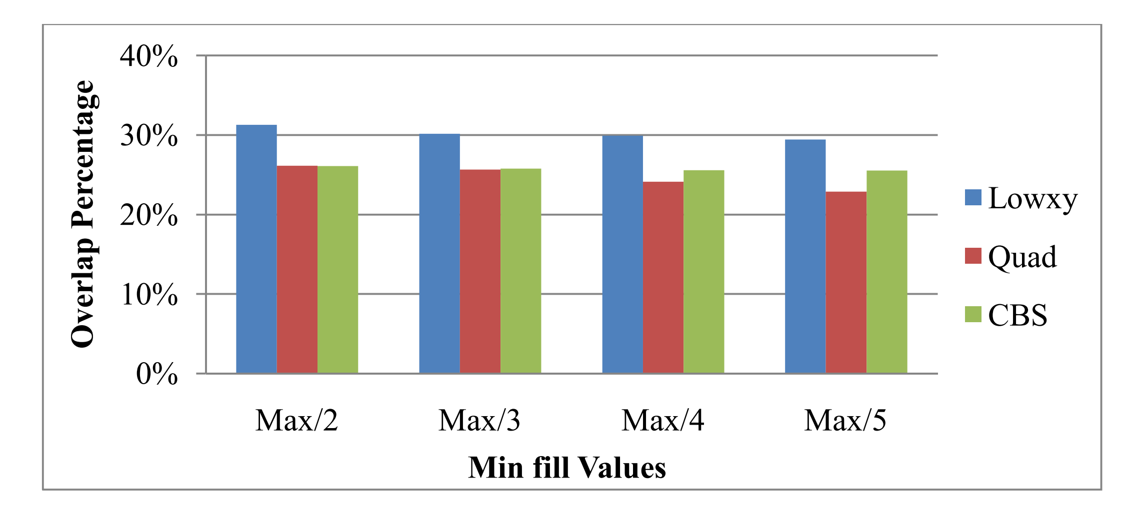


Figure 4.9: Overlap percentage between output nodes using Normal data file with 100000 entries, Max=50, with different Min fill values.

4.4 Index query tests

Query performance tests are measured by counting the number of disk accesses needed to fulfill a query; the I/O time in this situation is the dominant factor while the CPU time is negligible. We have used the intersection query type in running the performance test as it is the most used query type in the literature of node splitting testing (Theodoridis and Sellis, 1996).

By testing with different query sizes as a percentage of the space, the performance of the three algorithms can be measured and compared. The performance is measured as an average of ten uniformly distributed intersection queries for each query size. The following query sizes as a percentage of the test data world space were used in performing the tests: (0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 and 0.9). These different sizes cover small, medium and large query coverage of the space.

In this section we will present the query performance test results conducted using the files mentioned in the test data section. We will present the query tests performance comparison for indexes built by the different algorithms using the synthetic data files; Uniform and Normal, and for indexes built by the different algorithms using the real data files; CA and LB.

The comparison is presented as a ratio of number of disk accesses needed to fulfill the queries by each algorithm as being the performance testing criterion. For each query size, we ran 10 different randomly generated queries. The 10 runs were averaged then compared. Positive performance percentage ratio means better performance for the CBS algorithm, while negative performance percentage ratio means better performance for the other algorithm.

For all test data files used, we will present first the comparison for the Quad versus the CBS algorithms, then the comparison for the Lowxy versus CBS algorithms.

4.4.1 CBS algorithm versus Quad algorithm query test results: Comparison between the query test results of the CBS algorithm and the Quad algorithm shows that the CBS algorithm in most of the cases has a better performance than the Quad algorithm except for limited cases. We here present the test results for each data file used.

- **Uniform data file query performance test results:** We performed the query tests for indexes built using the Uniform data file with 100000 entries by both algorithms, while varying the Min fill value from Max/2 to Max/5. For each index, the query size was varied from 5% to 90% of the space, repeating the test 10 times for each query size then averaging the result. Table 4.8 lists the performance improvement percentage between the two algorithms. Positive performance percentage ratio means better performance for the CBS algorithm, while negative performance percentage ratio means better performance for the Quad algorithm.

Table 4.8: Performance improvement percentage between the CBS alg. and the Quad alg. for indexes created using Uniform data file with 100.000, Max=50 for different query sizes and different Min fill sizes.

Query Size	Min=Max/2	Min=Max/3	Min=Max/4	Min=Max/5
5%	0.02	-0.02	0.01	0.04
10%	0.02	-0.02	0.02	0.03
20%	0.02	0.01	0.06	0.07
30%	0.01	0.01	0.06	0.09
40%	0.02	0.02	0.08	0.10
50%	0.01	0.03	0.09	0.11
60%	0.00	0.03	0.09	0.11
70%	0.00	0.04	0.10	0.12
80%	0.00	0.05	0.10	0.13
90%	0.00	0.05	0.10	0.13

The CBS algorithm in most of the cases has a same or a better performance than the Quad algorithm. The only case where the Quad algorithm has a better performance is when Min Fill value = Max/3 and query size is 10% or less. Both algorithms show the same performance when Min fill = Max/2 and query size is 60% or more. In all other cases the CBS algorithm has better performance over the Quad algorithm. The percentage of improvement in performance ranges from 1.0% up to 13.0%. The percentage of improvement in performance gets better when the query size increase and Min fill value decrease. Figure 4.10 presents a comparison of the Performance improvement percentage between the CBS algorithm and the Quad algorithm for the different Min fill values and query sizes of table 4.8.

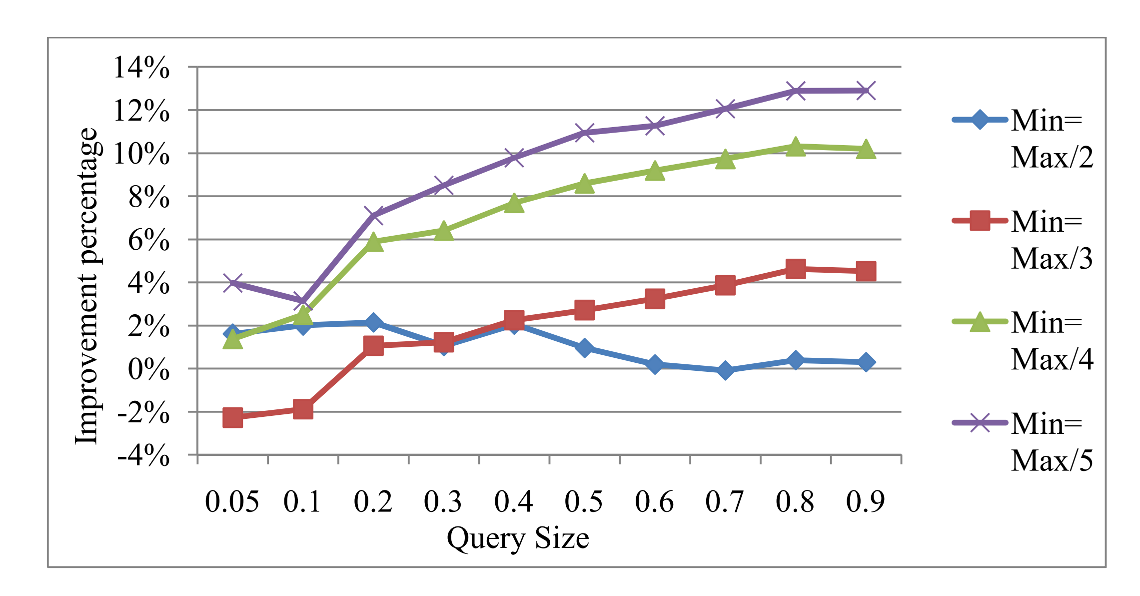


Figure 4.10: Performance improvement percentages between the CBS alg. and the Quad alg. for index created using Uniform data file with 100000 for different query sizes and different Min fill sizes

- **Normal data file query performance test results**: The above intersection query tests were repeated using the same settings to test indexes build by the two algorithms using the Normal data file with 100000 entries. Table 4.9 lists the performance improvement percentage between the CBS algorithm and the Quad algorithm for indexes created for different query sizes and different Min fill values.

Table 4.9: Performance improvement percentage between the CBS alg. and the Quad alg. for indexes created using Normal data file with 100000 entries, Max=50 for different query sizes and different Min fill sizes.

Query Size	Min=Max/2	Min=Max/3	Min=Max/4	Min=Max/5
5%	-0.05	-0.03	0.03	0.16
10%	-0.03	-0.01	0.07	0.17
20%	-0.02	0.01	0.09	0.20
30%	-0.02	0.01	0.09	0.21
40%	-0.02	0.01	0.10	0.22
50%	-0.02	0.02	0.10	0.22
60%	-0.02	0.02	0.11	0.23
70%	-0.01	0.02	0.11	0.23
80%	-0.01	0.02	0.11	0.23
90%	-0.01	0.02	0.11	0.23

By comparing the results for the CBS algorithm and the Quad algorithm in table 4.9, it shows that the CBS algorithm in most of the cases has a better performance than the Quad algorithm except for the case when Min fill value is Max/2 for all query sizes and the case when Min fill value is Max/3 and query size is less than 20%. All other cases show a performance improvement for the CBS algorithm over the Quad algorithm especially when Min fill is Max/4 and Max/5 where the improvement ratio is significant, the performance improvement ratio reaches 23% when Min fill is Max/5

and query size is greater than 50%. Figure 4.11 presents a comparison of the performance improvement percentage between the CBS algorithm and the Quad algorithm for the different Min fill values and query sizes of table 4.9.

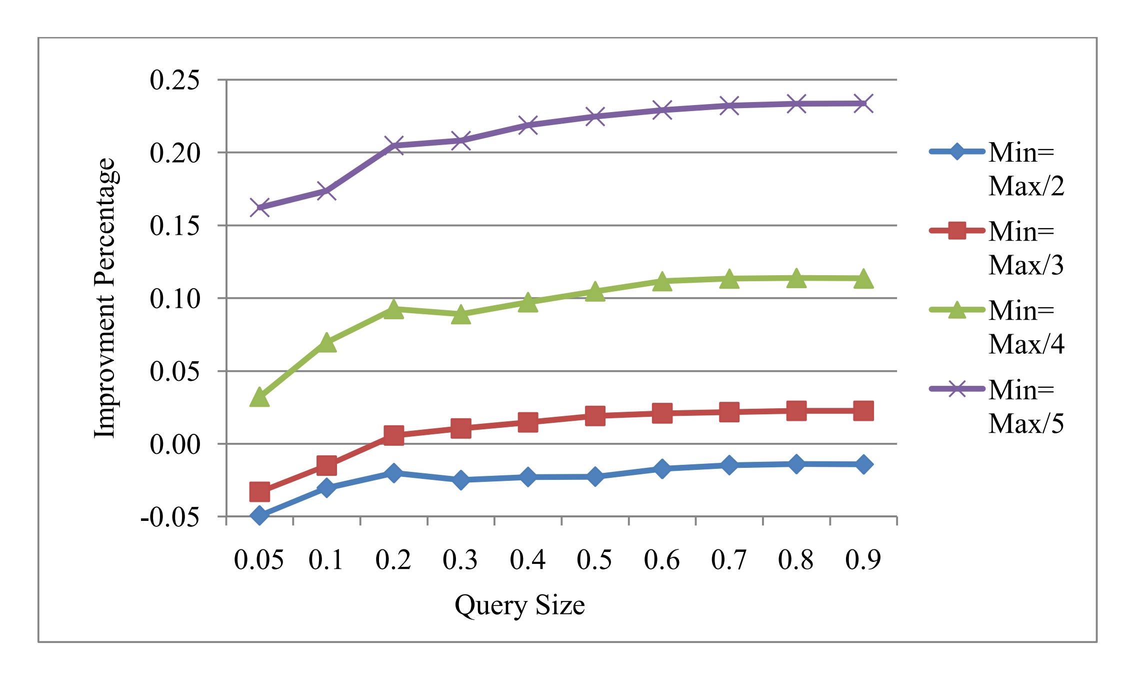


Figure 4.11: Performance improvement percentages between the CBS alg. and the Quad alg. for indexes created using Normal data file with 100000 entries for different query sizes and different Min fill values.

- **CA data file query performance test results:** The same intersection query tests were repeated using the same settings to test the performance of indexes built by the two algorithms using the CA data file with 98451entries which represents the MBRs of streams (poly lines) of California. Table 4.10 lists the performance improvement percentage between the CBS algorithm and the Quad algorithm for indexes created using CA data file when Max=50, for different query sizes and different Min fill values.

Table 4.10: Performance improvement percentage between the CBS alg. and the Quad alg. for index created using CA data file for different query sizes and different Min fill sizes.

Query Size	Min=Max/2	Min=Max/3	Min=Max/4	Min=Max/5
5%	-0.01	0.02	0.05	0.07
10%	0.00	0.01	0.03	0.08
20%	0.00	0.02	0.05	0.09
30%	0.00	0.02	0.05	0.09
40%	0.00	0.02	0.06	0.10
50%	0.00	0.02	0.06	0.10
60%	0.00	0.02	0.06	0.10
70%	0.00	0.02	0.06	0.10
80%	0.00	0.02	0.06	0.10
90%	0.00	0.03	0.07	0.11

By comparing the results for the CBS algorithm and the Quad algorithm, it shows that the CBS algorithm in most of the cases has a better performance than the Quad algorithm, except for the case when Min fill value is Max/2 for all query sizes where the two algorithms performance almost the same. All other cases show a performance improvement for the CBS algorithm over the Quad algorithm especially when Min fill value increases and the query size also increases. The performance improvement ratio is 10% when Min fill is Max/5 and query size is greater than 30%. Figure 4.12 presents the performance improvement percentage between the CBS algorithm and the Quad algorithm for indexes using the CA data file for different query sizes created and different Min fill values.

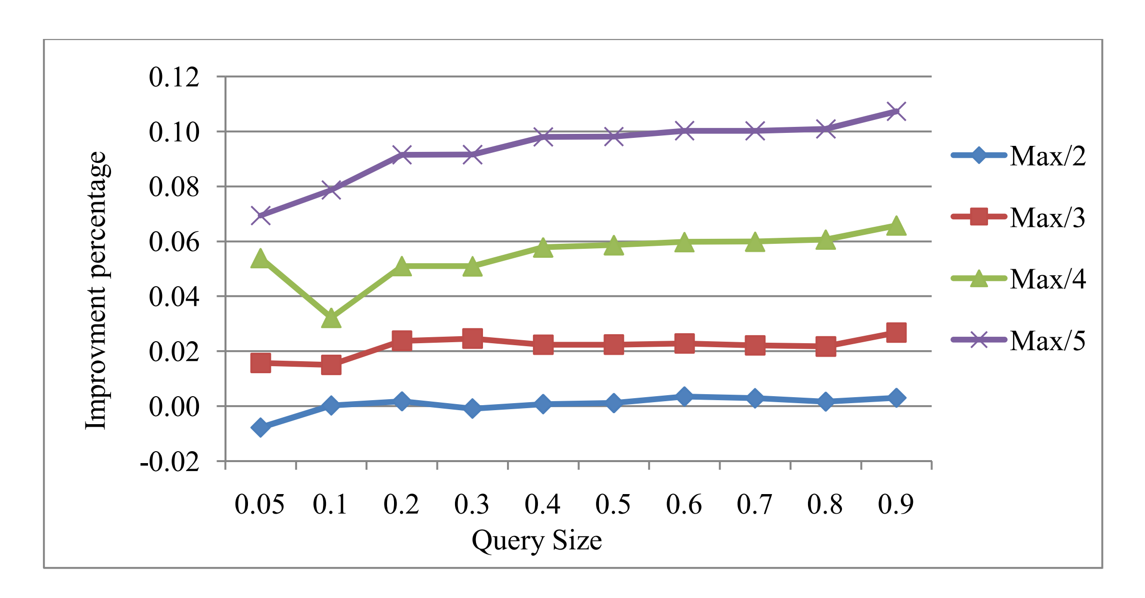


Figure 4.12: Performance improvement percentages between the CBS alg. and the Quad alg. for indexes created using the CA data file for different query sizes and different Min fill values.

LB data file query performance test results: The above intersection query tests were repeated using the same settings to test indexes build by the two algorithms using the LB data file with 53145entries represents the Long Beach county roads. Table 4.11 lists the performance improvement percentage between the CBS algorithm and the Quad algorithm using LB data file for different query sizes and different Min fill sizes.

Table 4.11: Performance improvement percentage between CBS and Quad alg. using LB data file, Max=50 for different query sizes and different Min fill values.

Query Size	Min=Max/2	Min=Max/3	Min=Max/4	Min=Max/5
5%	-0.06	-0.03	0.04	0.04
10%	-0.03	0.02	0.05	0.08
20%	-0.03	0.02	0.07	0.09
30%	-0.02	0.03	0.08	0.11
40%	-0.02	0.03	0.08	0.12
50%	-0.01	0.04	0.08	0.12
60%	-0.01	0.03	0.08	0.12
70%	0.00	0.04	0.09	0.12
80%	0.00	0.04	0.09	0.12
90%	0.00	0.04	0.09	0.12

By comparing the results in table 4.11 for the CBS algorithm and the Quad algorithm, it shows that the CBS algorithm in most of the cases has a better performance than the Quad algorithm except for the case when Min fill value is Max/2 with query size less than 50% and the case when Min fill is Max/3 and query size is 5%. In these two cases the Quad algorithm performance is better than the CBS algorithm. All other cases show a good performance improvement for the CBS algorithm over the Quad algorithm especially when Min fill value increases and the query size also increases. The performance improvement ratio is 12% when Min fill is Max/5 and query size is greater than 30%. Figure 4.16 presents the performance improvement percentage of Table 4.11.

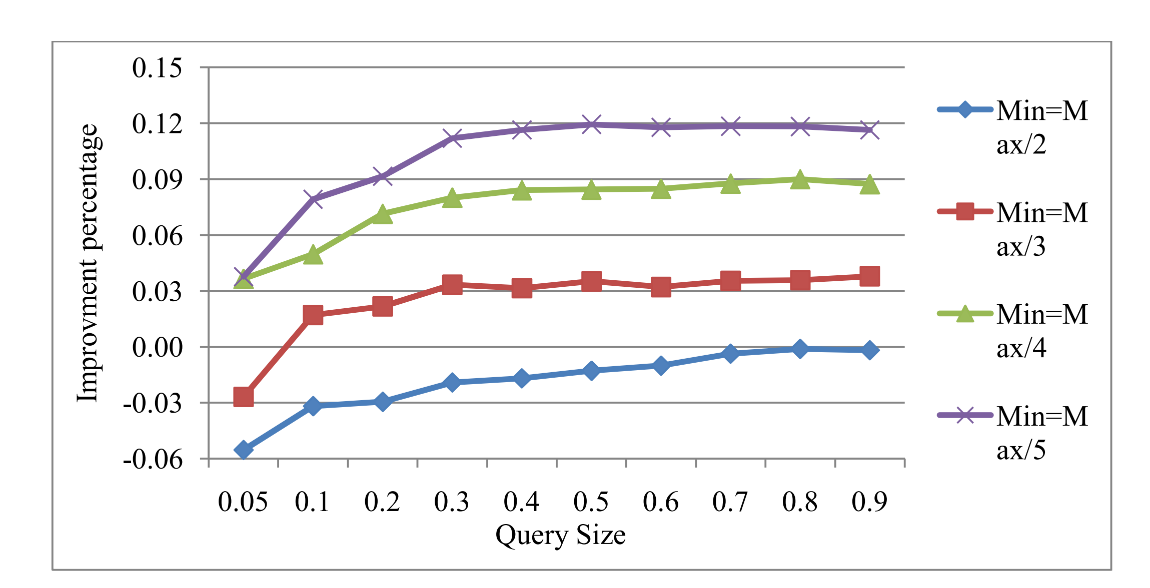


Figure 4.13: Performance improvement percentages between the CBS alg. and the Quad alg. for indexes created using the LB data file for different query sizes and different Min fill values

4.4.2 CBS algorithm versus Lowxy algorithm query test results: Comparison of the query test results for the CBS algorithm versus the Lowxy algorithm shows that the CBS algorithm in all the cases has a better performance than the Lowxy algorithm. The percentage of improvement in performance ranges from 0% up to 37%. The improvement ratio is high when using small query sizes. When query size increases, improvement in performance ratio decrease. Here we present the results for each data

file used. Positive performance percentage ratio means better performance for the CBS algorithm.

- **Uniform data file query performance test results:** We performed the query tests for indexes built using the Uniform data file with 100000 entries by the two algorithms, varying the Min fill value from Max/2 to Max/5. For each index, the query size was varied from 5% to 90% of the space, while repeating the test 10 times for each query size then averaging the result. Table 4.12 lists the performance improvement percentage between the two algorithms.

Table 4.12: Performance improvement percentage between the CBS alg. and the Lowxy alg. for indexes created using Uniform data file with 100.000, Max=50, for different query sizes and different Min fill sizes.

Query Size	Min=Max/2	Min=Max/3	Min=Max/4	Min=Max/5
5%	0.37	0.29	0.31	0.36
10%	0.27	0.19	0.17	0.22
20%	0.20	0.14	0.14	0.18
30%	0.15	0.11	0.12	0.14
40%	0.14	0.10	0.11	0.12
50%	0.11	0.08	0.10	0.09
60%	0.09	0.07	0.09	0.08
70%	0.07	0.06	0.08	0.07
80%	0.06	0.05	0.06	0.06
90%	0.04	0.04	0.04	0.04

By comparing the results of the CBS algorithm and the Lowxy algorithm for the Uniform data file in table 4.12, it shows that the CBS algorithm has a better performance than the Lowxy algorithm for all Min fill values and query sizes. The improvement percentage is high when Min Fill values and query sizes less than 50%. The improvement percentage decrease when the Min fill value and the query size

increase. Figure 4.14 presents the performance improvement percentage between the CBS algorithm and the Lowxy algorithm for indexes created using Uniform data file.

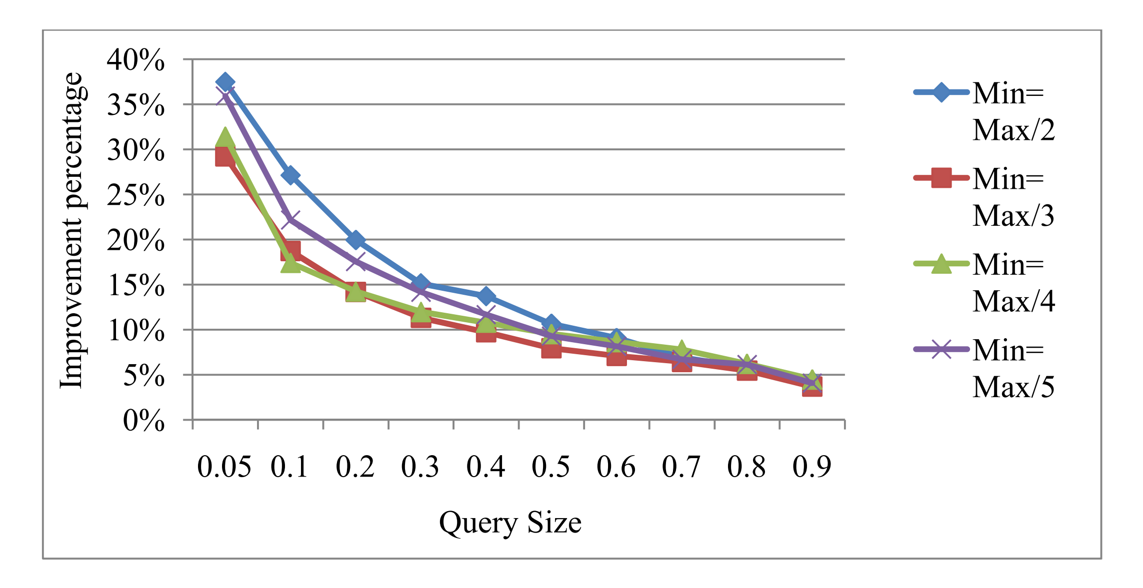


Figure 4.14: Performance improvement percentage between the CBS alg. and the Lowxy alg. for index created using Uniform data file with 100000 for different query sizes and different Min fill sizes.

- **Normal data file query performance test results:** The same intersection query tests were repeated using the same settings to test indexes build by the two algorithms using the Normal data file with 100000 entries. Table 4.13 lists the performance improvement percentage between the CBS algorithm and the Lowxy algorithm for indexes created for different query sizes and different Min fill values.

Table 4.13: Performance improvement percentage between the CBS alg. and the Lowxy alg. for indexes created using Normal data file with 100000 entries for different query sizes and different Min fill sizes.

Query Size	Min = Max/2	Min = Max/3	Min = Max/4	Min = Max/5
5%	0.10	0.12	0.13	0.15
10%	0.06	0.08	0.07	0.09
20%	0.05	0.05	0.05	0.07
30%	0.03	0.03	0.03	0.04
40%	0.01	0.02	0.01	0.02
50%	0.01	0.01	0.01	0.02
60%	0.01	0.01	0.01	0.01
70%	0.01	0.01	0.01	0.01
80%	0.01	0.01	0.00	0.01
90%	0.01	0.01	0.00	0.01

By comparing the results of the CBS algorithm and the Lowxy algorithm for the Normal data file in table 4.13, it shows that the CBS algorithm has a better performance than the Lowxy algorithm for small query sizes with all Min fill values. For medium and large query sizes, the two algorithms performance is almost the same. Figure 4.15 presents the performance improvement percentage between the CBS algorithm and the Lowxy algorithm for indexes created using Normal data.

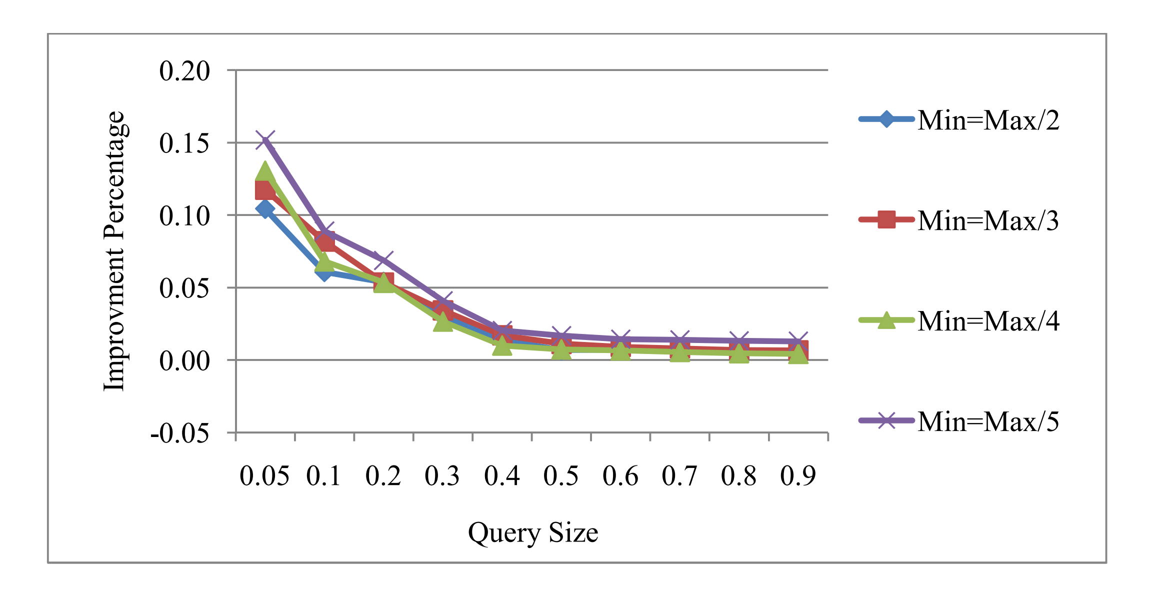


Figure 4.15: Performance improvement percentages between the CBS alg. and the Lowxy alg. for indexes created using Normal data file with 100000 entries for different query sizes and different Min fill values.

- **CA data file query performance test results:** The same abovementioned intersection query tests were repeated using the same settings to test indexes build by the two algorithms using the CA data file with 98451 entries represents the MBRs of streams (poly lines) of California. Table 4.14 lists the Performance improvement percentage between the CBS algorithm and the Quad algorithm for indexes created using CA data file when Max=50, for different query sizes and different Min fill sizes.

Table 4.14: Performance improvement percentage between the CBS and Lowxy for indexes created using CA data file for different query sizes and Min Fill sizes.

Query Size	Min=Max/2	Min=Max/3	Min=Max/4	Min=Max/5
5%	0.06	0.05	0.03	0.01
10%	0.05	0.04	0.00	0.03
20%	0.02	0.04	-0.01	0.02
30%	0.02	0.03	0.00	0.01
40%	0.01	0.03	0.00	0.01
50%	0.02	0.02	0.00	0.01
60%	0.01	0.02	0.00	0.01
70%	0.01	0.02	0.00	0.01
80%	0.01	0.02	0.00	0.00
90%	0.00	0.02	0.00	0.00

By comparing the results for the CBS algorithm and the Lowxy algorithm for the CA data file in table 4.14, it shows that the CBS algorithm has a better performance than the Lowxy algorithm for all Min fill value and small query sizes. For medium and large query sizes, the two algorithms almost have the same performance. Figure 4.16 presents the performance improvement percentage between the CBS algorithm and the Lowxy algorithm for indexes created using CA data file.

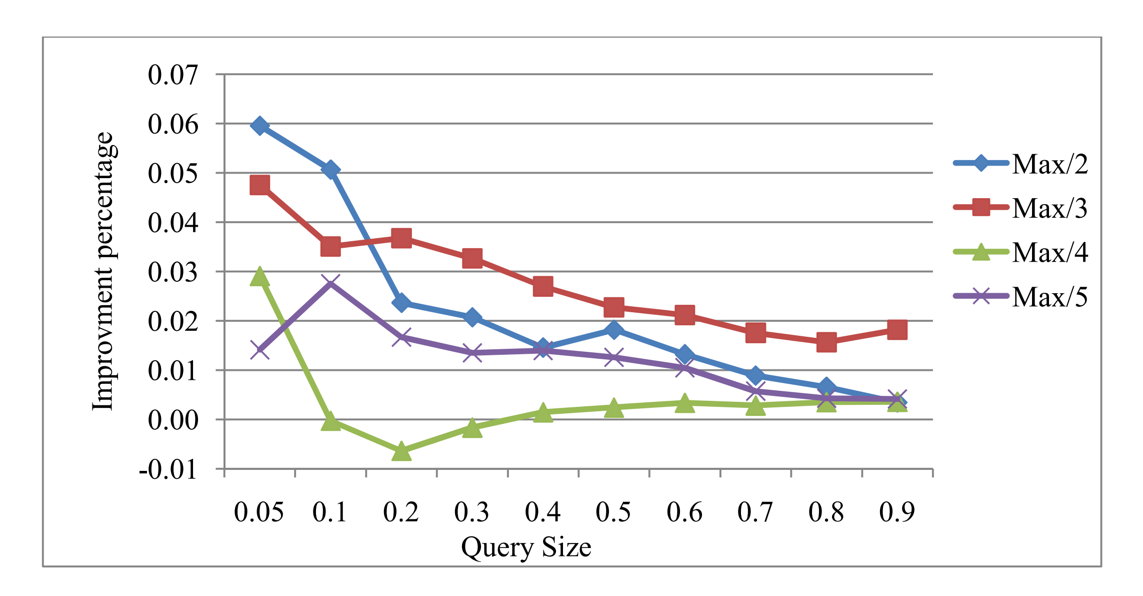


Figure 4.16: Performance improvement percentages between the CBS alg. and the Lowxy alg. for indexes created using the CA data file for different query sizes and different Min fill values.

- **LB data file query performance test results:** The above intersection query tests were repeated using the same settings to test indexes built by the two algorithms using the LB data file with 53145entries represents the Long Beach county roads. Table 4.15 lists the performance improvement percentage between the CBS algorithm and the Quad algorithm using LB data file for different query sizes and different Min fill sizes.

Table 4.15: Performance improvement percentage between CBS alg. and Lowxy alg. using LB data file, Max=50 for different query sizes and Min Fill values.

Query Size	Min=Max/2	Min=Max/3	Min=Max/4	Min=Max/5
5%	0.14	0.17	0.10	0.14
10%	0.10	0.12	0.09	0.14
20%	0.05	0.07	0.04	0.09
30%	0.03	0.05	0.03	0.08
40%	0.02	0.03	0.02	0.06
50%	0.02	0.03	0.01	0.06
60%	0.02	0.02	0.01	0.05
70%	0.02	0.02	0.01	0.05
80%	0.02	0.01	0.01	0.04
90%	0.01	0.02	0.00	0.04

By comparing the results for the CBS algorithm and the Lowxy algorithm for the LB data file in table 4.15, it shows that the CBS algorithm has a better performance than the Lowxy algorithm when query size is less than 50%, for all Min fill value. For query sizes greater than 50% the two algorithms almost have the same performance. Figure 4.17 presents the performance improvement percentage between the CBS algorithm and the Quad algorithm for indexes created using LB data file for different query sizes and different Min fill sizes.

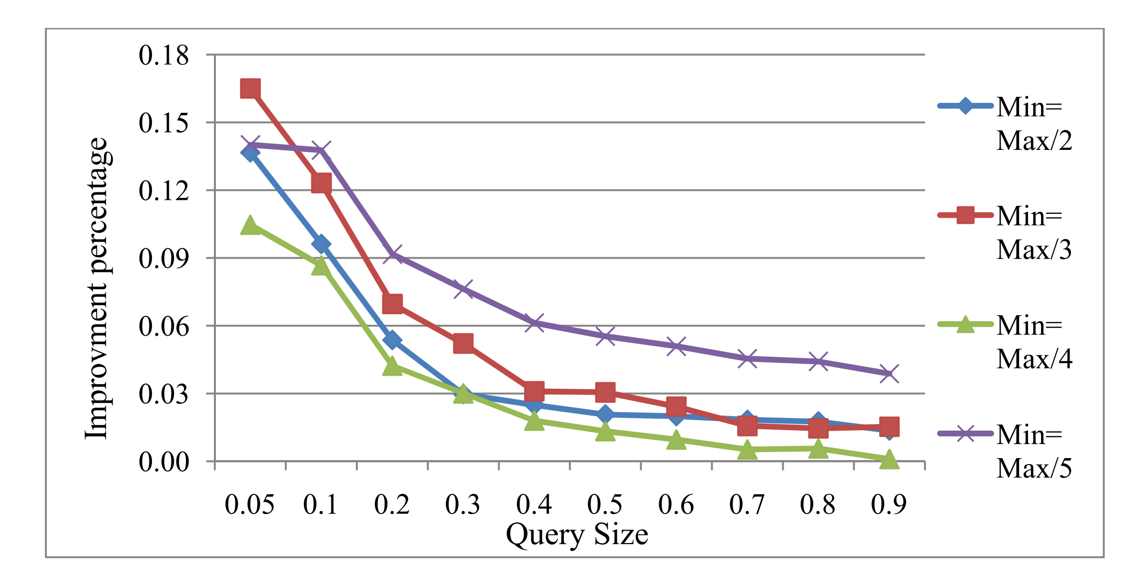


Figure 4.17: Performance improvement percentages between the CBS alg. and the Lowxy alg. for indexes created using the LB data file for different query sizes and different Min fill values

CHAPTER 5
CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

Indices are required for efficient access to spatial database systems. The R-tree spatial index structure is widely accepted and is used in many spatial applications. R-tree uses the MBR object approximation method to store the spatial data objects. Minimizing the overlap and the coverage of the index nodes' MBRs are crucial to the overall performance of the index. These criteria should be taken into account, especially during the process of splitting an over flown node.

Good splits produce an efficient R-tree structure which has a minimal height, minimal overlap between nodes and minimal coverage in each node. Many splitting algorithms were proposed in the literature to improve the R-tree's original splitting algorithms. Enhancing the full node splitting algorithm of the R-tree spatial index reduces the time needed to construct the index, increases the overall performance and enhances the distribution of data after splitting. Most importantly it will support a better query performance.

In this thesis, we presented a new improved method to split over flown nodes of the R-tree index structure called Corner Based Splitting (CBS) algorithm. The CBS algorithm exploits the properties of the rectangle shape to perform the splitting along one of the node's main axes. The CBS algorithm selects the splitting axis which will produce the most even split according to the number of objects, the least coverage area and overlap between the output nodes.

This is done by using the distance from each object center to the nearest corner to discover how objects are spread inside the node's MBR. A list of near objects for each corner is maintained, the count of objects in each list will determine which corners are

to be joined to produce the output nodes. If one of the resulting nodes is to receive fewer entries than the Min Fill constraint, a balancing mechanism is invoked to move the nearest objects to the splitting axis from the more filled node to the less filled node. The proposed method works for 2 or higher dimensional indices as well.

The experiments to test the CBS algorithm were done using both synthetic and real data test files, while changing the disk page size and the Min Fill value. It showed a superior performance for the CBS algorithm over two other splitting algorithms: The original R-tree Quad splitting algorithm and one of the most recent proposed algorithms; the Lowxy splitting algorithm.

The CBS algorithm outperforms both algorithms in the index creation time, overlap ratio and total coverage area. The data retrieval tests using intersection query while using different query sizes showed that the CBS algorithm needed less disk accesses (I/O) - in most of the cases – than both algorithms.

The improvement percentage of the CBS algorithm over the Quad algorithm increases when the query size and the Min Fill value are increased. The improvement percentage for the CBS algorithm over the Quad algorithm reaches up to 23%. The improvement percentage of the CBS algorithm over the Lowxy algorithm increases when the query size and the Min Fill are decreased. The improvement percentage for the CBS algorithm over the Lowxy algorithm reaches up to 37%.

5.2 Future Work

In all R-tree spatial index splitting algorithms, the main objective is to enhance the overall performance of the index by reducing the overlap and coverage areas. Although the CBS algorithm shows good performance compared to the other two algorithms in tests performed using 2-dimetional data, testing with higher dimensional data will verify the CBS algorithm ability to handle splits in such dimensions while preserving the same

performance. In addition, performing tests using query types other than the intersection query type, such as the window queries and containment queries, are also important to ensure the effectiveness of the CBS algorithm.

Testing the CBS algorithm against some other splitting algorithms such as the R*, R+ and the New Linear algorithms will demonstrate the strengths and weaknesses of the proposed algorithm.

REFERENCES

Al-Badarneh, A. Yaseen, Q. and Hmeidi, I. (2010), A new enhancement to the R-tree node splitting, **Journal of Information Science**, vol. 36 (1), pp: 3-18

Ang C. and Tan T., (1997), New linear node splitting algorithm for R-trees, **Proceedings of the International Symposium on Advances in Spatial Databases Conference**, pp: 339–349.

Bayer, R. (1997), **The universal B-Tree for multidimensional Indexing: General Concepts**, WWCA '97. Tsukuba, Japan, LNCS, Springer Verlag.

Beckmann N., Kriegel H., Schneider R. and Seeger B. (1990), The R*-tree: An efficient and robust method for points and rectangles, **Proceedings of the ACM SIGMOD Conference, pp:** 322–331.

Berchtold, S., Keim, D., and Kriegel H. (1996), The X-tree: An index structure for high-dimensional data, In **Proceedings of the International Conference on Very Large Databases (VLDB)**, pp: 28-39.

Bentley, J. L. (1975), "Multidimensional Binary Search Trees Used For Associative Searching," Communications of the ACM, 18(9), pp: 509-517,

Böhm, C. Berchtold, S. and Keim, D. (2001), Searching in High-Dimensional Spaces-Index Structures for Improving the Performance of Multimedia Databases, **ACM Computing Surveys**, Vol. 33(3), pp. 322–373.

Comer, D. (1979), **The Ubiquitous B-Tree,** ACM Computing Surveys, Vol. 11(2), pp: 121-137.

Freeston, M. (1995), A general solution of the *n*-dimensional B-tree problem, **Proceedings of the ACM SIGMOD International Conference on Management of Data,** pp: 80–91.

Fu, Y. Teng, J. and Subramanya S. (2002). Node splitting algorithms in tree-structured high-dimensional indexes for similarity search. In *Proceedings of the 2002 ACM symposium on Applied computing* **(SAC '02)**, ACM, New York, NY, USA, pp: 766-770.

Gaede, V., and Günther, O. (1997), Multidimensional access methods, **ACM Computing Surveys (CSUR)**, Vol. 30(2), pp: 170-231.

Garcia, R., Lopez, M., and Leutenegger, S. (1998), On optimal Node Splitting for R-trees, **Proceedings of the 24th VLBD Conference**, New York, USA.

Güting, R.H., and Schneider, M. (1995), Realm-based spatial data types: The ROSE algebra. VLDB Journal, 4, pp: 100-143.

Güting R.H. (1994), An Introduction to Spatial Database Systems**. VLDB Journal** 3 (4), pp: 357-399.

Guttman, A. (1984), R-Trees, a Dynamic Index Structure for Spatial Searching. **Proceedings of ACM SIGMOD Conference**, pp: 47-57.

Huang, P., Lin, P. and Lin, H. (2001), Optimizing Storage Utilization in R-tree Dynamic Index Structure for Spatial Databases, **Journal of Systems and Software**, Vol.55, pp: 291-299.

Hutflesz, A., H.W. Six, and P. Widmayer (1990), The R-file: An efficient access structure for proximity queries. In **Proc. 6th IEEE Int. Conf. on Data Eng**, pp.372-379.

Katayama, N. and Satoh, S. (1997), **The SR-tree: An index structure for high dimensional nearest neighbor queries**, *SIGMOD Record, Vol.* 26(2), pp: 369-380.

Kamel, I. and Faloutsos, C. (1994), Hilbert R-tree: An Improved R-tree using Fractals, **VLDB**, pp: 500-509.

Kriegel, H.P., M. Schiwietz, R. Schneider, and B.Seeger (1990) "Performance comparison of point and spatial access methods." In A. Buchmann, O. Gunther, T.R. Smith, and Y.F. Wang, **Design and Implementation of Large Spatial Database Systems**, Number 409 in LNCS, Berlin/Heidelberg/New York, pp: 89-114.

Lawder J. and King P., (2000), Using Space-Filling Curves for Multi-dimensional Indexing. In: **Advances in Databases, 17th British National Conference on Databases (BNCOD 17)***,* vol. 1832, Lecture Notes in Computer Science, pp: 20-35.

Liu, Y., Fang, J., Han, C. (2009), A new R-tree node splitting algorithm using MBR partition policy, **17th International Conference on Geoinformatics**, pp:1-6

Lomet D. and Salzberg B. (1990), The hB-tree: A multi-attribute indexing method with good guaranteed performance, **ACM Transactions on Database Systems**, 15(4), pp: 625-658.

Lomet, D. (1992): A Review of Recent Work on Multi-attribute Access Methods **SIGMOD RECORD,** Vol. 21(3), pp: 56-63.

Manolopoulos, Y., Nanopoulos A., Papadopoulos A.N. and Theodoridis Y. (2006), **R-trees: Theory and Applications**, USA, Springer-Verlag.

Nievergelt, J., H. Hinterberger, and K. Sevcik (1984), The grid file: An adaptable, symmetric multikey file structure, **ACM Transactions on Database Systems (9)**, pp: 38-71.

Nievergelt, J. (1989), 7±2 criteria for assessing and comparing spatial data structures. In **Design and Implementation of Large Spatial Database Systems**, A. Buchmann, O. Gunther, T. R. Smith, and Y.-F. Wang, Eds., LNCS 409, Springer-Verlag, Berlin/Heidelberg/New York, pp: 3–27.

Open GIS consortium: Open GIS simple features specification for SQL (Revision 1.1), http://www.opengis.org/techno/specs.htm.

Papadias D, Sellis T, Theodoridis Y, and Egenhofer M. (1995), Topological Relations in the World of Minimum Bounding Rectangles: A Study with R-trees. **ACM SIGMOD' 95**.

Robinson, J. T. (1981), “The K-D-B-tree: A Search Structure For Large Multidimensional Dynamic Indexes,” In Proc. ACM SIGMOD International Conference on Management of Data, pp: 10-18,

Samet, H. (1984), **The quadtree and related hierarchical data structures**, ACM Computing Surveys. Vol. 16(2),pp: 187-260.

Samet, H. and Webber, R. (1985), Storing a collection of polygons using quadtrees, **ACM Trans. Graph.** Vol. 4(3), pp: 182–222.

Samet, H. (1995), Spatial data structures. In **Modern database systems**, Won Kim (Ed.). ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp: 361-385.

Samet, H. (2004), Object-based and image-based object representations, *ACM Computing Surveys,* Vol 36(2), pp: 159–217.

Schneider, M. (1997), **Spatial Data Types for Database Systems - Finite Resolution Geometry for Geographic Information Systems**, LNCS 1288, Springer Verlag.

Seeger, B., and Kriegel, H. (1988), Techniques for design and implementation of efficient spatial access methods, **In Proceedings of the Fourteenth International Conference on Very Large Data Bases**, pp: 360–371.

Seeger, B. and Kriegel H. (1990), The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base Systems, **Proceedings of the 16th VLDB Conference**, pp: 590-601,

Sellis, T., Roussopoulos, N. and Faloutsos, C. (1987), The R+-tree: A dynamic index for multidimensional objects, **Proceedings of the VLDB Conference,** pp: 507–518.

Shekhar, S. Chawla, S. Ravada, S. Fetterer, A. Liu, X. Lu, C. (1999), Spatial Databases - Accomplishments and Research Needs, In **IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING**, VOL. 11 (1).

Silberschatz, A., Korth, H. and Sudarshan, S. (2006), **DATABASE SYSTEM CONCEPTS,** (5TH ed). Singapore: McGrawHill.

Six, H. and Widmayer, P. (1988), Spatial searching in geometric databases. In **Proceedings of the Fourth IEEE International Conference on Data Engineering**, pp: 496–503.

Skopal, T. Hoksza, D. and Pokorny, J.(2007), Construction of Tree-Based Indexes for Level-Contiguous Buffering Support, In R. Kotagiri et al. (Eds.): **DASFAA 2007**, LNCS 4443, Springer-Verlag Berlin Heidelberg. Pp: 361–373

Theodoridis, Y and Sellis, T., (1996), A model for the prediction of R-tree performance. In *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* (PODS '96). ACM, New York, NY, USA, pp: 161-171.

Thiodoridis, Y., R-Tree Portal, www.rtreepotal.org, 2001, (last date accessed May, 2011).

Wang, L., Yu, S., Chen, F. (2010), A new solution of node splitting to the R Tree algorithm, **International Conference on Intelligent Control and Information Processing (ICICIP)**, pp: 611-614.

White , D. and Jain, R. (1996), Similarity indexing with the SS-tree, In **Proc. 12th IEEE Int. Conf. on Data Engineering**, New Orleans, Louisiana, pp: 516-523.

Yaseen, Q. (2006), **A New approach of node splitting in R-tree**, Unpublished Master thesis, Jordan University of Science and Technology, Irbid, Jordan.

التحسين على طريقة فصل العقد الطرفية في الشكل التنظيمي الشجري من نوع (R)

إعداد

عصام يوسف النسور

المشرف

الدكتور عزام سليط

ملخـــــص

يستخدم الشكل التنظيمي الشجري (R-tree) في بناء قواعد البيانات الخاصة بالتطبيقات التي توظف استخدام الأجسام والأشكال متعددة الابعاد للتخزين والإسترجاع ولإجراء الإستعلامات المكانية عليها، ومثال هذه التطبيقات أنظمة المعلومات الجغرافية وبرمجيات التصميم. تتم عملية التخزين في (R-tree) من خلال حفظ الشكل ضمن مستوعب ذو شكلٍ رباعيٍ يمثل الحد الأدني للمساحة اللازمة للتخزين. الأشكال القريبة تخزن ضمن عقدة واحدة بحيث لا يتجاوز مجموعها الحد الأقصى الذي تستوعبه الصفحة في جهاز التخزين. عند إضافة شكل جديد لعقدة مليئة فإنه يتوجب فصل العقدة الى عقدتين وتقسيم مجموع الاشكال بينهما. كلما قلت مساحة التغطية للعقدة وقل التداخل بين العقد المتجاورة كلما زادت كفاءة التخزين والاسترجاع.

يهدف البحث الى اقتراح طريقة جديدة لفصل عقد الشكل الشجري (R-tree) تسمى خوارزمية الفصل بإستخدام الزوايا (CBS). تسعى الخوارزمية المقترحة لزيادة كفاءة (R-tree) من خلال تقليل مساحات التغطية والتداخل بين العقد. تقوم الخوارزمية بإستغلال مزايا الأشكال الرباعية بحيث تفصل العقد على امتداد أحد المحورين الرئيسين للشكل الرباعي الذي يضمن افضل توزيع للأشكال على العقد الناتجة. يتم تقرير محور الفصل بواسطة احتساب المسافة بين مركز كل شكل داخل العقدة إلى زاوية العقدة الأقرب لذلك الشكل. الأشكال القريبة من الزوايا المتباعدة تسند لعقد مختلفة.

التجارب على الخوارزمية المقترحة أستخدم فيها ملفات اصطناعية وحقيقية وتمت المقارنة مع اثنتين من الخوارزميات الموجودة. دلت النتائج أن للخوارزمية المقترحة الأفضلية في اختصار الوقت اللازم لبناء الشكل التنظيمي، وتقليل مساحات التغطية والتداخل، كما استدعت الاستعلامات المكانية للخوارزمية المقترحة عدداً أقل من عمليات القراءة من جهاز التخزين وبنسبة تحسين تصل الى 23% افضل من خوارزمية (Quad) وبنسبة تصل الى 37% أفضل من خوارزمية (Lowxy).